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:	D23
Deliverable Title :	Grasp simulator and prediction engine and interaction simulation with soft contacts
Type:	PU/PP
Authors	B. León, G. Puche, J. Felip, A. Morales, S. Ulbrich, T. Asfour R. Dillmann, S. Moisio, and P. Korkealaakso, Jeannette Bohg, Danica Kragic
Contributing Partners	UJI, KIT, LUT, KTH

Contractual Date of Delivery to the EC:	28-02-2011
Actual Date of Delivery to the EC:	28-02-2011

Contents

Chapter 1

Executive Summary

Deliverable 23 describes third year work within work-package WP6 "Introspection and Prediction through Simulation". Previous Deliverables (D9 and D16) from WP6 were focused on the development of the simulation environment. The third one still accounts for newer developments, but also includes applications of the simulator with predictive purposes. According to the Technical Annex, D23 presents activities connected to Tasks 3.1, 3.2, 3.3. The objectives of these are defined as:

- **[Task 6.1]: Implementation of the engine core architecture and the representation standards.** The architecture of the application and design details will be determined according to the precise description of the needs of the present and future users. This definition will deliver a specification of the internal data of the simulator, and the interfaces to the basic modules that will use it.
- **[Task 6.2]: Development of the basic modules.** The different modules of the simulator will be developed following the next sequence, though some of them can be developed simultaneously: Core modules (internal data management, communication, etc), and basic visualisation; robot oriented plug-ins: sensor simulation, advanced hand models, etc; object collision detection; static modelling (friction, contacts); dynamic modelling.
- **[Task 6.3]: Implementation of the reasoning/prediction engine**. The simulator developed in task 6.2 will be the base to make hypothesis and predictions about the world. This task will be responsible of implementing an engine that performs such operations. It will keep the representation and modelling of the objects involved in the interactions and will also produce predictions of the sensor perceptions (tactile, force and visual) to be obtained when the tasks are executed by real hardware. This engine is critical to the arousing of surprise since the difference between the predicted perceptions and real outputs is the first level of it.

The work in this Deliverable is related to **Milestone 8:** "Implementation of hybrid controllers for on-line adaptive primitive grounding; evaluation in the simulator and on experimental platforms."

The progress in WP6 is presented briefly below, and in more detail in the appendix containing attached scientific publications and reports.

- Attachment A describes the whole work carried on the development of the simulator, OpenGRASP, and describes several applications of the toolkit on grasping applications.
- Attachment B presents the description of the soft contact model which has been included in the distribution of OpenGRASP. It discusses a detailed implementation of it and describes how can it be used in order to simulate a real tactile sensor. The performance of the simulated sensor is validated and it is demonstrated how it can be integrated on the simulation of a complete robot grasping system.
- Attachment C describes two setups in which the simulator and real robots are integrated in order to complete a manipulation task. On these two setups robots capture partial views of objects, and a

simulator module predicts the whole shape of them, and plan grasps hypothesis according to these predictions. Hypothesis are evaluated, and the best one is chosen, and executed, on the real robot.

The work presented is remarkable in two aspects. First, it shows an integration of hardware, simulator and several modules in a unique framework: object shape prediction, grasp hypothesis generation, and collision-free path planning. Second, it describes a case of a complete implementation of the predict-act-perceive loop.

Appendix A

Attached papers

A Beatriz León, Stefan Ulbrich, Rosen Diankov, Gustavo Puche, Markus Przybylski, Antonio Morales, Tamim Asfour, Sami Moisio, Jeannette Bohg, James Kuffner, and Rüdiger Dillmann. OpenGRASP: A toolkit for robot grasping simulation. In *SIMPAR '10: Proceedings of the 2st International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, Darmstatd, Germany. November 2010. Best Paper Award.

B Sami Moisio, Beartiz León, Pasi Korkealaakso, and Antonio Morales. Model of Tactile Sensors Using Soft Contacts and its Application in Robot Grasping Simulation. Submitted to *IEEE Transactions on Robotics, special issue on Robotic Sense of Touch.*

C Jeannette Bohg, Matthew Johnson-Roberson, Beatriz León, Javier Felip, Xavi Gratal, Niklas Bergström, Danica Kragic, and Antonio Morales. Mind the gap - robotic grasping under incomplete observation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Shangai, China, May 2011. To appear.

OpenGRASP: A Toolkit for Robot Grasping Simulation

Beatriz León[1], Stefan Ulbrich[2], Rosen Diankov[5], Gustavo Puche[1], Markus Przybylski[2], Antonio Morales[1], Tamim Asfour[2], Sami Moisio[3], Jeannette Bohg[4], James Kuffner[5], and Rüdiger Dillmann[2] ⋆

[1] Universitat Jaume I, Robotic Intelligence Laboratory, Castellón, Spain,
{len,puche,morales}@icc.uji.es
[2] Karlsruhe Institute of Technology, Institute for Anthropomatics, Karlsruhe, Germany,
{ulbrich,przybyls,asfour,dillmann}@ira.uka.de
[3] Lappeenranta University of Technology, Centre of Computational Engineering and Integrated Design, Finland,
smoisio@lut.fi
[4] Royal Institute of Technology, Computer Vision and Active Vision Laboratory, Stockholm, Sweden,
bohg@csc.kth.se
[5] Carnegie Mellon University, Institute for Robotics, USA,
{rdiankov,kuffner}@cs.cmu.edu

Abstract. Simulation is essential for different robotic research fields such as mobile robotics, motion planning and grasp planning. For grasping in particular, 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 environments and robots, including the modeling of actuators, sensors and contacts. In this paper, we present a new simulation toolkit for grasping and dexterous manipulation called *OpenGRASP* addressing those aspects in addition to 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. In addition, a designated editor has been created for the generation and migration of such models. We demonstrate the current state of OpenGRASP's development and its application in a grasp evaluation environment.

Keywords: software toolkit, grasping simulation, robot modeling

1 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

⋆ The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° 215821.

industrial manipulator manufacturers offer simulation packages accompanying their robotic 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 the analysis of behaviours and performance beforehand. In robotics research, simulators have an important role in the development and demonstration of algorithms and techniques in areas such as path planning, grasp planning, mobile robot navigation, and others. There are several reasons for the use of robot simulations. First, they allow exhaustive testing and tuning of mechanisms and algorithms under varying environmental conditions. Second, they avoid the use, or misuse, and wearing of complex and expensive robot systems. Finally, simulation software is cheaper than real hardware.

Often, simulation tools used to support research are specifically developed for particular experiments. However, there have been some successful attempts to develop general robot simulators specifically for mobile robotics.

Stage and Gazebo are respectively 2D and 3D simulator back-ends for Player [1, 4], which is a widely used free software robot interface. Gazebo [2] in particular, implements a 3D multi-robot simulator which includes dynamics for outdoor environments. It implements several robot models, actuators and sensors. USARSim [5] has a similar functionality. It is a free mobile robot simulator based on a gaming engine. Microsoft provides its Robotics Studio [3], a framework for robot programming that includes a visual simulation environment. OpenHRP [7] is an open software platform with various modules for humanoid robot systems such as a dynamics simulator, a view simulator, motion controllers and motion planners. OpenHRP is integrated with CORBA, with each module, including the dynamics simulator implemented as a CORBA server. Commercial options include Webots [6], a product which has been widely successful in educational settings.

The variety of simulation tools for robotic grasping is rather limited. The most renowned and prominent one is *GraspIt!*, a grasping simulation environment [9]. *GraspIt!* includes models of several popular robot hands, implements the dynamics of contacting bodies, includes a grasp planner for a Barrett hand and has recently included a simple model of the human hand. However, *GraspIt!* has several limitations. Its rather monolithic and less modular architecture makes it difficult to improve, add functionality and integrate with other tools and frameworks. In addition, it does not provide a convenient Application Programming Interface (API), which allows script programming. Furthermore, it does not include sensor simulation.

Another existing and publicly available software framework is OpenRAVE, the Open Robotics and Animation Virtual Environment [10]. It has been designed as an open architecture targeting a simple integration of simulation, visualization, planning, scripting and control of robot systems. It has a modular design, which allows its extension and further development by other users. Regarding robot grasping simulation, it provides similar functionality to GraspIt! and various path planning components. It provides the models of several robot

arms and hands and allows the integration of new ones. It also enables the development of virtual controllers for such models.

In this paper we introduce a simulation toolkit specifically focused on robot grasping and manipulation. The principal design issues have been extensibility, interoperability and public availability (see Sec. 2). The core of the toolkit is an improved version of OpenRAVE, which has been enhanced with a Robot Editor and the adoption of the COLLADA file format [12] and Physics Abstraction Layer (PAL) [11] to enable standardization and flexibility (see Sec. 3). Finally, several applications which demonstrate the utility of the toolkit are presented (see Sec. 4).

This work is the result of a team funded by the European Commission within the project GRASP [21]. Within the project the toolkit is used as the main tool for reasoning and prediction of object properties as well as task constraints of manipulation and grasping activities executed by robots. Thus, it is a key tool for constructing a world model and understanding the robot environment.

2 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 a practical and user point of view, it should include the models of the most popular robot hands, and provide the possibility of easily creating and adding new ones. Furthermore, it should provide realistic simulations of real actuators and sensors, which would enable the use of the same API as in their real counterparts. Regarding sensors, a grasping simulator has to provide simulations of specific grasping sensors, such as force/torque, contact and tactile. 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 enables on the one hand, an easy extension by both developers and users and on the other hand, the integration with commonly-used robotics software frameworks. In order to increase its opportunities of being adopted 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 to import/export robot and object models to/from standard representations.

To our best knowledge, none of the existing simulation tools and software packages fulfill all of these requirements. Therefore, we present a software toolkit for grasping simulation *OpenGRASP* [8], which is built on top of OpenRAVE to meet the requirements discussed above.

3 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-available software packages and standards.

After a wide review of existing simulators, physics engines, 3D render engines, and CAD 3D modelers, we concluded that OpenRAVE is the tool that most closely meets our requirements. So our efforts have focus on improving and extending OpenRAVE capabilities and features towards the realization of an advanced grasping simulator. In addition, we developed a robot editor allowing to create and modify new robot models. In the following sections we describe these extensions in more detail.

3.1 OpenRAVE Architecture

OpenRAVE 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).

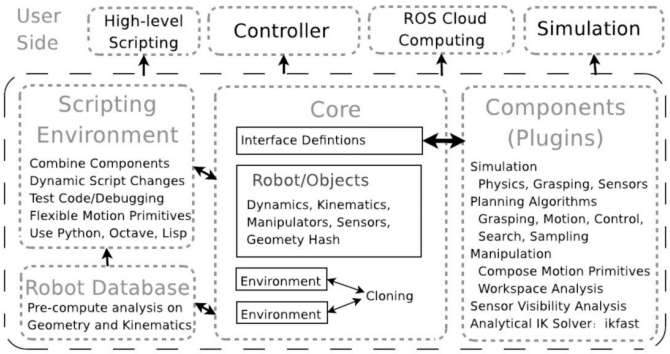


Fig. 1. OpenRAVE Architecture (Picture reproduced from [22])

The Scripting Layer enables 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 enable 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 the creation of 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 to recompile the core. Following this design, different

plugins can be created for components 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 the Robot Operating System (ROS) [30].

A GUI can be optionally attached to provide a 3D visualization of the environment. It periodically queries the core to update the world's view and allows the user to change the position of the objects in the scene. As viewers are provided through plugins, a single OpenRAVE instance can allow multiple viewers to communicate with multiple copies of the environment. A more detailed description of the OpenRAVE architecture can be found in [10, 22].

Although many plugins are already implemented to provide basic functionality, the current grasp simulation functionality 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 help to compare and improve simulation performance.
- Implementation of a standard plugin interface for a basic actuator and implementations of different motors. This would allow us to accurately simulate the robot's arm and hand articulations.

We have taken these considerations into account in our toolkit design. We have developed two new sensor plugins to be used mainly for anthropomorphic robot hands. One is a tactile sensor, commonly used in finger tips such as in the Barrett hand, which detects and calculates the forces on the predefined sensory area and returns 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.

We have created a model of the Schunk PG70 parallel jaw gripper using the tactile sensor plugin to simulate the Weiss Robotics sensor (DSA 9205) attached to each finger. We have also provided the Gripper Sensor plugin which returns the distance between fingers, their current and velocity. Finally, a model of the gripper actuator was also included which can control the velocity of the fingers and the maximum current applied.

In order to use different physics engines, we have implemented a plugin which makes use of the physics abstraction layer (PAL), which is addressed in more detail in the following 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 3.4) 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.

3.2 Physics Simulation

Nowadays, there exist many available physics engines, both commercial and open-source. Some of them are high-precision engines that require higher computational power while 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, some rely on physical laws using constraint equations, and others use methods based on impulses.

None of these engines are perfect, they all have advantages and disadvantages which makes it very difficult to decide which one to use for a simulator. It basically depends on what we want to simulate in addition to what the application of the simulator will be.

The Physics Abstraction Layer (PAL) [11] is a software package created by Adrian Boing that saves us from having to decide at the start what physics engine to use. This layer provides an interface to a number of different physics engines allowing us to dynamically switch between them. This functionality adds incredible flexibility to our simulator offering us the possibility to, depending on our specific environment and use, decide which engine would provide us the best performance [14]. Using their interface, it is also possible test and compare our own engines with existing ones.

The OpenRAVE Physics Engine interface allows the simulator to run using different engines. It also has 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 basic version of OpenRAVE only offers the ODE (Open Dynamics Engine) implementation 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.

3.3 COLLADA File Format for Robot Models

For the storage of models of robot manipulators and hands, we were looking for an open, extensible and already widely accepted file format that supports the definition of at least kinematics and dynamics. This is necessary in order to enable the exchange of robot models between supporting applications, leading to greater flexibility in the selection of appropriate tools. Another important aspect was the ability to convert to and from other formats. Among the large variety of file formats for 3D models, there are only a few that are both public domain and not limited to storing only geometric information. Typically, a simulator environment does not only rely on geometric structures but also, for instance, on information about dynamics, kinematics, sensors and actuators of the robot.

Its acceptance as an industry standard and its wide distribution (3D Studio, Blender, OSG, OGRE, Sony, etc.), in addition to a clear and extensible design, led to the choice of COLLADA [12] as the preferred file format for the simulator. In addition, there are open source frameworks available that facilitate integration into new applications.

Since version 1.5, the standard contains many useful constructs dedicated to describing kinematic chains and dynamics that can be used directly for the description of robot models. COLLADA is an XML-based file format that enables and encourages developers to extend the specification to their needs without having to violate the underlying schema definition.

In order to support specific robot features like sensors and actuators, we have used this mechanism to extend COLLADA partially using the original OpenRAVE file definition. These additions are specific to the simulator and are hidden to all other applications so that compatibility remains guaranteed. So far, basic support for COLLADA import and export has been included in the simulator.

3.4 Robot Editor

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

- **Geometric modeling:** The creation of new robots models requires a tool that excels in modeling of the geometric components (i.e., meshes).
- **Semantic modeling:** Even more important is the ability to allow the description of semantic properties, such as definitions of kinematic chains, sensors and actuators, or even specify algorithms.
- **Dynamics modeling:** Another necessary 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 modeling tool needs to be capable of processing these formats and converting them into the COLLADA standard. GraspIt! files in particular, being an already widely-used standard with many conform models available, should be readily usable by the simulator.

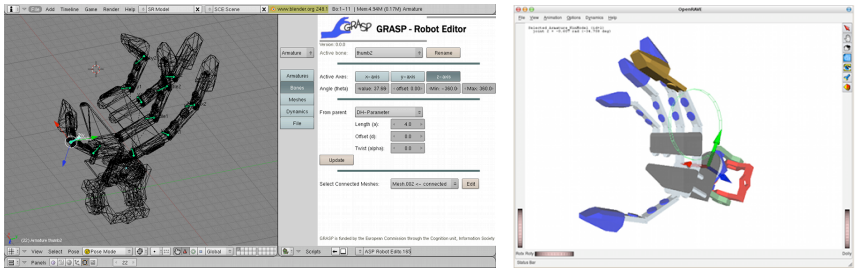
To our knowledge, there is no existing solution openly available that could meet all these requirements. Therefore, we decided to develop a new modeling tool based on available open source software. The conceptual design of the Robot Editor hence relies on two techniques: on the one hand the open data format COLLADA and on the other hand on the open source project Blender [15]. Blender is a very versatile, powerful and extensible 3D editor that has been chosen because of its convenient 3D modeling capabilities and the built-in support for many CAD file formats and rigid body kinematics. Furthermore, it can be easily extended via a Python scripting interface and offers high-quality ray tracing.

Blender itself, however, lacks the functionality and the appropriate interface for the convenient definition of robot components. In addition, conversions

between certain file formats need to be improved or implemented, namely the import of the COLLADA format and GraspIt! robot models.

The scripting interface mechanism mentioned above allowed us to build the modeling tool on top of Blender. On the scripting level, one gains access to all relevant data structures. The robot model can be augmented by the required data structures and conserved within the native file format. The scripting mechanism also allows the creation of user-interface that is highly specialized for use in robotics (see Fig. 2(a)). For instance, you can define kinematics via Denavit-Hartenberg parameters. In the long run, the Robot Editor will provide interfaces for essential robotics algorithms, such as 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, in the future, will be useful for developers' planning algorithms.

In the current Blender version (2.49), COLLADA support [17] is limited to documents in the older specification 1.4 which excludes the newly introduced kinematics and dynamics. The additional data required by the simulator also needs to be included in the resulting document. This led to the further development of COLLADA compatibility which now enables the Robot Editor to create valid documents suitable for simulation. Fig.2(a) shows a functional model of the Karlsruhe anthropomorphic hand [16] modified using the Robot Editor and Fig.2(b) the resulting COLLADA file loaded into the simulator .



(a) (b)

Fig. 2. Modeling the Karlsruhe anthropomorphic robot hand. a) The user interface of the Robot Editor and b) Screenshot of the complete model in the simulator.

Robot Models As stated in Section 2, it is of great importance to have models of the most popular robot hands included in the toolkit. The modeling capabilities of the Robot Editor already enable quick and easy creation of new models. So far, a selection of robot hands has been transformed into COLLADA 1.5 for use within the simulator (see Fig. 3). In addition to these new models, there are various models available in the older XML file format which is still supported.

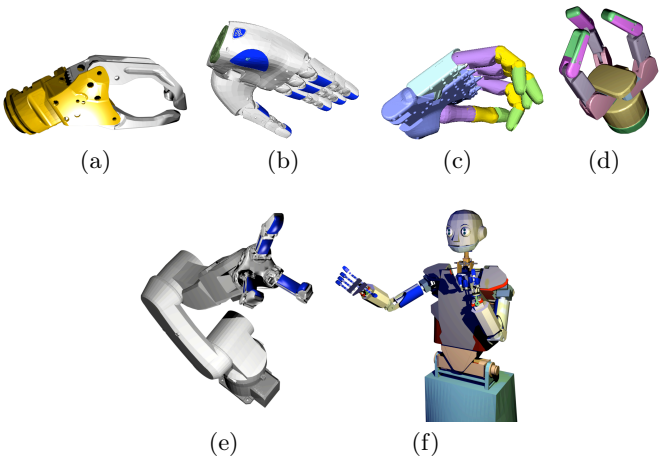


Fig. 3. Different robot models generated with the Robot Editor (ray-traced images). (a) A myoelectric upper extremity prostheses of Otto Bock, the Schunk (b)SAH and (c)SDH hands, (d) the Shadow hand, (e) the Barrett hand and the Kuka KR5 sixx R850 arm, and (f) the humanoid robot ARMAR-III.

4 Applications

4.1 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 4 shows the grasp simulation process by analyzing the contact points between the robot and the target object. Recently Przybylski et al. [26] used OpenGRASP to develop a new grasp planning method based on the Medial Axis of objects to reduce the search space of candidate grasps. The algorithm exploits structural and symmetry information contained in the Medial Axis in order to reduce the search space for promising candidate grasps.

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 an 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, robots developed in our RobotEditor can manipulate everyday objects.

Grasping Known Objects Using the planning and grasping capabilities of the simulator, we have successfully grasped known objects in the real world with the robotic platform consisting of the Karlsruhe Humanoid Head (see [27, 28]) and a Kuka KR5 sixx R850 arm equipped with a Schunk Dextrous hand 2.0.

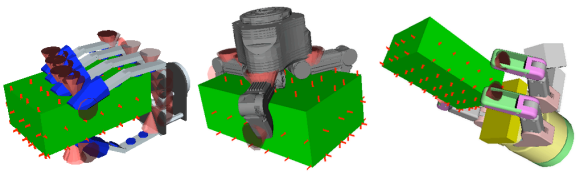
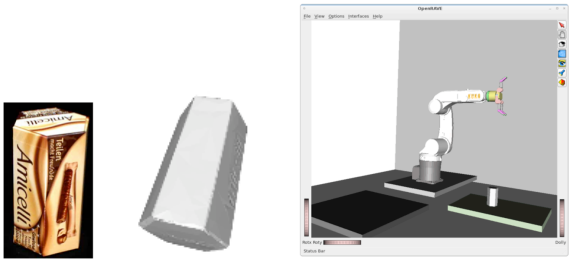


Fig. 4. Grasping simulated for several robot hands.

A complete description of known objects is provided by the KIT object model database [29]. The object description consists of geometry described by a mesh and physical properties like mass. Using the grasp simulation process, different grasp hypotheses are tried out off-line. Only those that result on force closure are then saved to a data base. Using an active vision system [23] in combination with rigid point set registration [24], the robot can explore the current scene and recognize where and in which pose these known objects are (see an example in Fig. 5). This information is sent to OpenGRASP which reproduces the state of the environment. Using the planning plugins, the selected force closure grasps are executed. The ones that collide with the environment or that are not reachable given the current configuration of the robot are discarded.



a. b. c.

Fig. 5. Grasping a known object: a) real object, b) mesh geometry and c) example of a scene visualization after its recognition

Grasping Unknown Objects We have also used OpenGRASP to manipulate unknown objects with the above mentioned robotic platform. This task is more complex since no information about the objects is available. Thus, the grasp simulation process cannot be executed off-line. The robot uses the above mentioned active vision system [23] to explore the scene and segment unknown objects. From the reconstructed stereo point cloud, a mesh is created as an approximation of the object's geometry. This mesh and its location are sent to the simulator which uses an available library for decomposing it into a reduced

number of convex hulls to simplify its collision detection. Given this mesh, there is an unlimited number of grasp configurations that can be applied to the object and tested for force closure. For reducing the search space and thereby online processing time, we apply a grasp point detection method [25] to the segmented object. The simulator then tries only grasp configurations with the fingers of the robotic hand being aligned with the grasp points. Those that are in force closure are saved in a database before being executed with the planner to check for collisions and reachability. An example of the successful grasping of an unknown object is shown in Fig. 6.

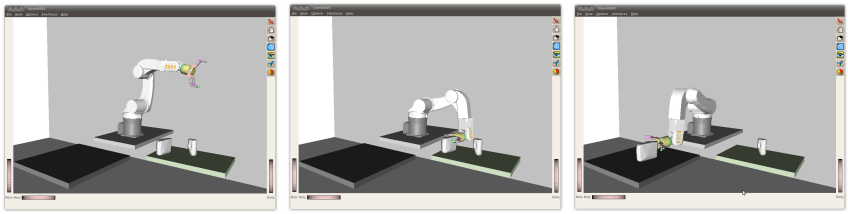


Fig. 6. A successful grasp of an unknown object with the Kuka KR5 R850 arm and a Barret hand.

5 Conclusion and Future Work

In this paper we have presented a fully operational simulation toolkit for robot grasping and manipulation. Its main design principles are extensibility, interoperability and public availability. In its development we have used existing and widely-available components to ensure its standardization and easy adoption. We have also emphasised providing additional tools and features that provide users with a fast start to enhance utility through features such as the robot editor based on Blender, COLLADA file format, a Physics Abstraction Library, and models of existing robot hands. The utility of OpenGRASP is demonstrated through a series of applications cases.

In future, we are looking to improve the toolkit in three directions. First a novel contact modelling for soft contacts and body deformation is being implemented and validated. It will be included as a library to allow the modelling of complex contact interactions. Second, OpenGRASP is being extended to offer full support for ROS thus offering a common control interface for simulated and real robots. Finally, several interfaces are being created in order to offer the user several options to visualize and record the simulated systems.

References

1. Gerkey, B., Vaughan, R., Howard, A.: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: 11th International Conference on Ad-

vanced Robotics (ICAR 2003), pp. 317–323. Coimbra, Portugal (2003)
2. Gazebo, http://playerstage.sourceforge.net/index.php?src=gazebo
3. Microsoft Robotics Studio, http://msdn.microsoft.com/robotics
4. The Player Project, http://playerstage.sourceforge.net/wiki/Main_Page
5. USARSim, http://sourceforge.net/projects/usarsim/
6. Webots, http://www.cyberbotics.com/
7. OpenHRP, http://www.is.aist.go.jp/humanoid/openhrp/
8. OpenGRASP, http://opengrasp.sourceforge.net/
9. Miller, A., Allen, P.: GraspIt!: A versatile simulator for robotic grasping. In: IEEE Robotics & Automation Magazine. vol. 11, pp. 110–122 (2004)
10. Diankov, R., Kuffner, J.: OpenRAVE: A Planning Architecture for Autonomous Robotics. Technical report, Robotics Institute, Pittsburgh, PA (2008)
11. PAL (Physics Abstraction Layer), http://www.adrianboeing.com/pal
12. COLLADA, http://collada.org
13. Khronos Group, http://www.khronos.org/
14. Boeing, A., Bräunl, T.: Evaluation of real-time physics simulation systems. In: 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia (GRAPHITE'07), pp. 281–288. Perth, Australia (2007)
15. Blender, http://Blender.org
16. Gaiser, I., Schulz, S., Kargov, A., Klosek, H., Bierbaum, A., Pylatiuk,C., Oberleand, R., Werner, T., Asfour, T., Bretthauer, G., Dillmann, R.: A new anthropomorphic robotic hand. In: IEEE/RAS International Conference on Humanoid Robots (Humanoids), pp. 418–422 (2008)
17. Illusoft: Blender Collada Plugin, http://colladablender.illusoft.com/cms/
18. Blender - Google Summer of Code 2009, http://www.blendernation.com/blender-google-summer-of-code-2009/
19. OpenCOLLADA, http://www.opencollada.org/
20. Otto Bock, http://www.ottobock.de/cps/rde/xchg/ob_de_de/hs.xsl/384.html
21. GRASP Project, http://www.csc.kth.se/grasp/
22. OpenRAVE website, http://openrave.programmingvision.com/
23. Björkmann, M., Kragic, D.: Active 3D scene segmentation and Detection of Unknown Objects. In: International Conference on Robotics and Automation (ICRA 2010), Anchorage, Alaska, USA, (2010)
24. Papazov, C. , Burschka, D.: Stochastic Optimization for Rigid Point Set Registration. In: Proceedings of the 5th International Symposium on Advances in Visual Computing (ISVC '09). pp. 1043–1054 (2009)
25. Richtsfeld, M., Vincze, M.: Grasping of Unknown Objects from a Table Top. In: ECCV Workshop on 'Vision in Action: Efficient strategies for cognitive agents in complex environments', Marseille, France (2008)
26. Przybylski, M., Asfour, T., Dillmann, R.: Unions of Balls for Shape Approximation in Robot Grasping. To appear in: IROS (2010)
27. Asfour, T., Regenstein, K., Azad, P., Schröder, J., Vahrenkamp, N., Dillmann, R.: ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control. IEEE/RAS International Conference on Humanoid Robots (Humanoids), pp. 169–175, (2006)
28. Asfour, T., Welke, K., Azad, P., Ude, A., Dillmann, R.: The Karlsruhe Humanoid Head. IEEE/RAS International Conference on Humanoid Robots (Humanoids), pp. 447–453, (2008)
29. Kasper, A., Becher, R., Steinhaus, P., Dillmann, R.: Developing and Analyzing Intuitive Modes for Interactive Object Modeling. International Conference on Multimodal Interfaces, (2007). See also http://i61www.ira.uka.de/ObjectModels)
30. ROS, http://www.ros.org/wiki/

Model of Tactile Sensors Using Soft Contacts and its Application in Robot Grasping Simulation

Sami Moisio, Beatriz León, *Member, IEEE*, Pasi Korkealaakso and Antonio Morales, *Member, IEEE*

***Abstract*—In the context of robot grasping and manipulation, realistic simulation requires an accurate modeling of contacts between bodies and, in a practical level, an accurate simulation of touch sensors. This paper addresses the problem of simulating a tactile sensor considering soft contacts and full friction description. Firstly, a complete theoretical model of contact is developed. It is based on a penalty (soft-contact) approach. It consists of a surface contact patch described by a mesh of contact elements. For each element, a full friction description is built considering stick-slip phenomena. On a further stage, the model is implemented and used to build a realistic simulation of a real tactile sensor. The performance of the simulated sensor is validated. It is also demonstrated how it can be integrated on the simulation of a complete robot grasping system.**

***Index Terms*—Force and Tactile Sensing, Contact Modelling, Grasping, Animation and Simulation.**

I. Introduction

ROBOTIC manipulators have been used extensively for many years in both research and commercial applications. These applications vary from usual industrial operations such as assembly, drilling and welding in the car manufacturing process, to more sophisticated tasks such as performing surgery, repairing a space station and assisting with house cleaning tasks.

Over the years, the change from structured scenarios to real environments has made the development of different sensors a priority to enable robots to cope with significant uncertainties.

Touch, combined with vision, are the main senses that allow humans to perform dexterous manipulation. For this reason, sensors that can retrieve tactile information have been developed in order to equip robot hands with such a sense (see [1], [2] for a review).

Tactile sensors can measure different properties of the objects they make contact with. They can be mechanical properties including pressure, normal and shear forces, torques, slip and vibrations, or other properties like temperature or moisture. In this study the mechanical properties of the sensor are covered.

The performance of the real tactile sensors developed until now is far from human sensing. Nevertheless, they have been used in robot manipulation in the last few years for different purposes including reactive robot control, collision detection and object recognition.

In reactive control, the robot has to cope with the inaccuracy of the vision systems when working in unstructured environments. In such cases, the robot should be able to archive a grasp, even if the pose of the object is not perfectly known, using tactile information [3]. Usually a tactile sensor placed on the robot's palm will report when a contact occurs while approaching the object. This contact point is assumed to be the position of the object and then used to recenter the object inside the hand [4]. Combining tactile information with vision and force feedback enables more complex manipulation tasks such as door handle grasping, door opening or grasping books from a bookshelf [5], [6]. Recent studies have also shown how tactile-sensing-based algorithms can be employed to detect and react to contacts encountered during the execution of a grasp [7] as well as how tactile sensor information can be used to infer knowledge about grasp stability [8].

In the field of object recognition, tactile sensors are used to explore the 3D shape of unknown objects and use their feedback to create or improve the object's model. Some properties of the object can be derived from the contact information such as stiffness, texture or friction coefficient [9]. The tactile information can enable a robot to identify objects from its observations [10], [11]. Using the tactile sensor matrix, a small imprint of the object can be taken and used to recognize surface features [12].

These various applications of tactile sensors show the importance of their use in robot manipulation. In this area, simulation is a major tool used to support research, adding both flexibility and reproducibility to the experiments. Having a tactile sensor model that replicates the behaviour of the real sensor will be of great benefit to the robotics community.

Robot simulation has great advantages over using real hardware. Changing or repeating a configuration with the real robot can be very time consuming but in simulation these situations are easily and efficiently solved, by changing the simulation parameters. In addition, the simulation model is not limited to existing hardware thus allowing easy changes like the sensors type or position. Another advantage of using a simulator is the availability of different information. The real hardware always requires an actual sensor but the simulator has all the information related to the virtual world available. This availability of information enables the construction of virtual sensors such as querying the distance to the nearest object from the manipulator's end effector. These virtual sensors offer great possibilities for research in grasping without being restricted by existing hardware. They might even offer new ideas for

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° 215821.

S. Moisio and P. Korkealaakso are with Centre of Computational Engineering and Integrated Design (CEID) at the Department of Machine Technology, Lappeenranta University of Technology, P.O. Box 20, 53851 Lappeenranta, Finland e-mail: {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 e-mail: {len,morales}@uji.es

Manuscript received August 20, 2010

constructing new types of hardware sensors.

Some simulation environments such as GraspIt! [13], [14] have been developed for the purposes of robot manipulation. However, in the most of the cases, robotic grasping simulations are solved using kinematics instead of dynamics. The problem is that the use of soft contacts in kinematic systems is a difficult task. This is due to the fact that they do not include any degree of freedom as the motion is carried out using forced motions and there is no need to solve contact or actuator forces. In addition, the use of kinematics usually results in incorrect object mass and inertial data being used in the simulations making the results incomparable to the real case. For this reason, it is preferred to use the dynamics simulation approach in the case of soft contacts where the contact forces can gradually take place over several time-steps. On the other hand, when the soft contacts are compared to non-penetrating contacts the collision detection results in major assumptions in terms of contact area as it does not exist any penetration between objects.

This study presents the development of a simulated tactile sensor element with the same physical properties as a real tactile sensor would have: compressibility, friction, etc. In order to create a model of the sensor dynamics three different areas were addressed: tactile sensor construction, modeling soft contacts and friction modeling. All these areas were combined to make a physical model of a tactile sensor. The sensor element type itself is universal and can be used to model any kind of tactile sensor. 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.

In order to test the proposed model, different experiments were conducted. First, the basic physical properties of the simulated tactile sensor were validated. Then, experiments on robot grasping were carried out by a robot hand grasping an object and by the corresponding model on the simulator performing the same actions. Furthermore, an experiment demonstrating how the simulated tactile sensor could be used in object recognition was also carried out.

The paper is organized as follows. The previous work on contact models and grasping in simulation is reviewed in Section II. The model proposed in this paper is detailed on Section III. An implementation of the theoretical model has been developed for an open-source simulator and is presented in Section IV. In Section V, experiments on robot grasping are presented, followed by an analysis of their results. Section VI concludes the paper with a summary, a discussion of the experiment's results and proposed future work.

II. Previous Work

This section first presents a review of the contact models used in simulation, explaining their advantages and disadvantages followed by the relevant work on simulation of robot grasping.

A. Contact Models

Contact models can be divided into three different categories: analytical, impulse and penalty methods. Rigid body assumption for collisions is used in the analytical [15] and in the impulse methods [16], [17] while continuous contact models are used in the penalty methods [18]–[20]. 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 contact between bodies is considered as a collision at a specific point in time and without needing to solve the contact forces, instead the change in the objects' velocities is applied directly to the bodies over one time-step. The method is fast and easy to implement but a problem arises with steady contacts in static configurations. Analytical methods are based on the use of constraints to handle contacts. In contrast to impulse-based methods, these methods are stable in steady contacts, however, due to simultaneous solving of all contacts, they are also computationally expensive. Penalty methods are 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 temporal nonlinear springdamper elements at the contact point. Based on the elasticity of the bodies in contact, the parameters of the spring-damper element can be defined using the Herzian contact theory [21].

Analytical and impulse methods give accurate descriptions for contacts and are often used when no interpenetrations are allowed between the contacting bodies. These methods also allow longer time-steps compared to penalty methods with stiff springs. However these methods lead to complicated equations 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 require 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 assumptions do not take into account small deformations during collisions, instead there occur instantaneous changes in velocities. For this reason, continuous contact models give more accurate descriptions of contact forces during the contact period.

Conventional penalty methods use only the deepest contact point for contact forces. In [22], a geometry-based approach is used in order to find exact contact areas of the polygons applying contact forces to multiple points. However, in most cases the algorithm is not efficient enough for real time simulation and it is highly dependent on the body geometry construction. One of the advantages in using penalty methods is the straightforward 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 guesses or assumptions since the objects are not allowed to penetrate and form a real contact surface. This in turn complicates various calculations such as holding torque around the contact area. For this reason a penalty method was chosen to be used in the sensor model developed in this paper.

B. Grasping in Simulation

Grasping in robotics using simulated tactile sensors is a new field of research. Some research has been done but the

developed methods are derived from existing non-penetrating contact models such as in [23]. In general, robot grasping simulations have traditionally been using kinematics instead of dynamics. This could be due to the fact that the simulations community has not cooperated extensively enough with the robotic grasping community or to the simple fact that simulated grasping is a very challenging problem. The most common simulation method for robot grasping has been the impulse method (GraspIt! [14], ODE [24], Bullet [25], etc.). The impulse method is a very effective method for simulating structures that form open kinematic chains which 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 [16], [17]. The problem occurs when a very light (Barrett hand fingertip is approximately 50 g) object tries to collide with a very heavy object (3 kg payload). Using a penalty method this mass dependency can be avoided in the collisions but the solution becomes sensitive to variables such as the time-step due to the stiffness of the system.

III. Tactile Sensor Model

The purpose of this work was to make a simulation model of a real tactile sensor, not just by emulating the function but by modeling the actual physical properties starting from the formation of an actual contact patch to including a full friction description.

The chosen method consists of using a contact patch with several contact elements in order to form the tactile sensor. Single tactile elements are used in order to determine collisions against other objects and to calculate the resulting collision forces. This provides an accurate and fast solution for solving the collision equations. Currently the number of contact points is equal to the number of tactile elements in the tactile sensor but in the future a single tactile element can be modified to include several contact elements in order to further increase the resolution of the sensor.

The contact forces are calculated on each contact point and are used by the simuator to grasp the object and for the tactile sensor feedback.

The tactile sensors produced by Weiss Robotics [26], specifically the DSA 9205 model, were used as the base line to validate our model. Before continuing, the working principle and limitations of these sensors are presented.

A. Weiss Tactile Sensors

Each sensor consists of 84 discrete sensor cells (texels) forming a homogeneous matrix of 6 x 14, which is able to detect an applied load profile. In order to do that, a resistive working principle detailed in [27] is used, which places a common electrode and sensing electrodes covered with an elastic rubber foam on each cell. The electrical resistance is measured as a function of the contact area between the electrode and the foam. When a force is applied to the cell, a deformation of the foam occurs, the contact area rises, thus lowering the electrical resistance of the material. This change in resistance is interpreted as an increasing load which is quantified and returned as an image of the applied pressure profile [28], [29].

These sensors are widely used by the robotics community because they were specifically developed to be used on robot hands. However, there are some known problems with their measurements [10], [30] that have made the validation of our tactile model a difficult task. The problems include the following:

- The sensors show significant hysteresis, even when texels are not loaded to their maximum value.
- Due to the layer of rubber foam, a considerable low-pass characteristic of the spatial impulse response can be observed.
- When exactly the same force is applied to two sensors, the texel values and their sum differ remarkably.
- Calibration of the sensor is difficult due to its working principle and initial values should be taking into account.

The values given out by these sensors are not actual forces or pressures. Instead they are called *intensities*. The simulated tactile sensor knows the exact forces in each sensor element which then have to be converted to tactile intensities. This is done using a linear conversion, based on a preliminary measurement on a real tactile sensor, by knowing that a certain load produces a certain tactile value sum. These conversions turn out to be very difficult to model due to the major differences in the tactile value sums between individual tactile sensors.

This also complicates the simulation model by adding a conversion to the model.

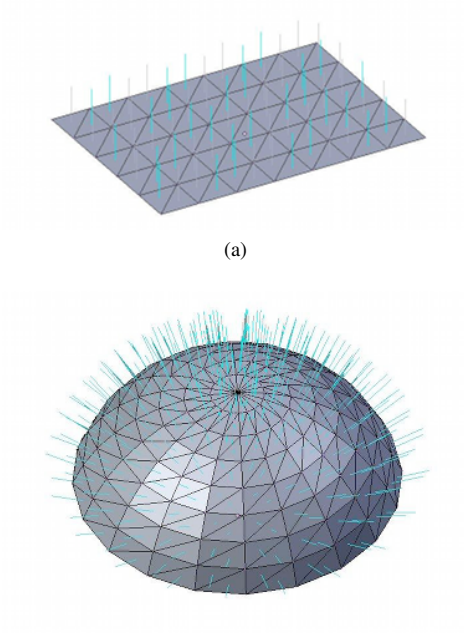
Knowing the limitations of the real tactile sensor helped us to create the simulated sensor model and to understand the results of the experiments presented in Section V which will illustrate the above problems.

In the following sections a detailed description of the tactile sensor model is presented.

B. Geometry-based tactile

The simulated tactile sensor element can be 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 it would therefore be difficult to describe it in order to form the tactile sensor array to encompass the finger tip. Using our model, the geometry of the simulated tactile sensor can be formed directly based on the finger tip geometry. In Fig. 1 two different variations of a tactile pad element array are presented: a simple grid (1a) and a spherical surface (1b). The arrows represent the normal directions of the different triangles.

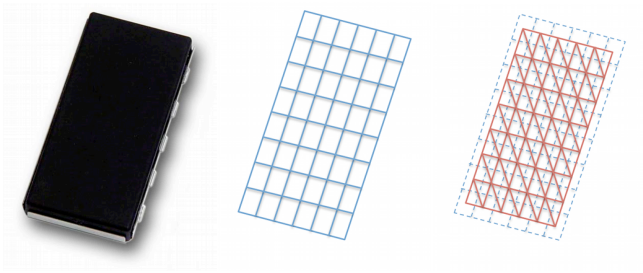
The array of the simulated tactile sensor elements is constructed using the vertices from the sensor geometry. In the case of a tactile sensor array with 6 rows and 8 columns, one



(a) (b)

Fig. 1. Example of tactile sensor geometries: (a) a simple grid and (b) a spherical surface.

would draw a 5x7 grid having 6x8 vertices to represent the centers of the simulated tactile elements (see Fig. 2).



(a) (b) (c)

Fig. 2. Example of a simulated tactile sensor construction: (a) real tactile sensor, (b) geometry of the sensor and (c) simulated tactile sensor elements

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's maximum penetration needs to be defined in the sensor parameters. It is 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 which creates a contact point. The forces calculated at this point are explained in the following section.

C. Contact Force Model

When a collision between the sensor element and an object occurs, the contact information (position, relative velocity, penetration, etc.) is used to calculate the force in a single tactile element.

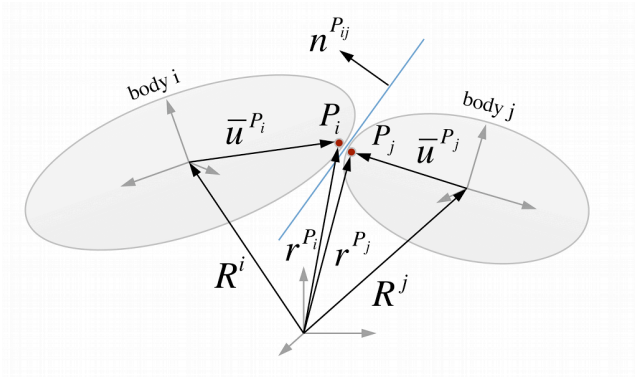


Fig. 3. Contact between bodies i and j.

The kinematics of the contact points between two bodies i and j can be described using knowledge of the geometries and states of the bodies (see Fig. 3).

The distance between contact points P_i and P_j can be written as follows:

$$s = r^{P_j} - r^{P_i} \tag{1}$$

where r^{P_i} and r^{P_j} are the position vectors of each contact point in the global reference frame. If R^i and R^j are defined as the center position vector of each body, the distance can be written as:

$$s = R^j + A^j \bar{u}^{P_j} - R^i - A^i \bar{u}^{P_i} \tag{2}$$

where A^i and A^j are rotation matrices from the body reference frame to the global reference frame and $\bar{u}^{P_i}$ and $\bar{u}^{P_j}$ are the position vectors of the contact points within the body reference frames.

The relative velocity between the contact points can be calculated by:

$$\dot{s} = \dot{R}^j + \tilde{\omega}^j A^j \bar{u}^{P_j} - \dot{R}^i - \tilde{\omega}^i A^i \bar{u}^{P_i} \tag{3}$$

where $\dot{R}^i$ and $\dot{R}^j$ are the velocity vectors of bodies i and j, and $\tilde{\omega}^i$ and $\tilde{\omega}^j$ are skew-symmetric matrices of the angular velocities.

By defining a contact plane between the bodies as the tangential plane to the normal of the bodies, the distance in the normal direction can be written as:

$$d = s^T n^{P_{ij}} \tag{4}$$

where s^T is the transpose vector of s and $n^{P_{ij}}$ is the normal vector of the contact plane. Accordingly, the velocity in the direction of the normal of the contact plane can be written as:

$$\dot{d} = \dot{s}^T n^{P_{ij}} \tag{5}$$

The relative velocity in the tangential direction of the contact plane can be obtained as follows:

$$\dot{s}_t = \dot{s} - \dot{d} n^{P_{ij}} \tag{6}$$

Contact forces are described using the soft contact approach which allows small penetration between contacting bodies taking into account local deformations.

On each contact point, the contact force (F_C) can be written as:

$$F_C = F_n + F_t \tag{7}$$

where F_n is the normal force produced by the soft contact and F_t is the tangential force represented by friction.

In its simplest form, the contact force in the normal direction of the plane (F_n) can be written as a linear spring-damper element:

$$F_n = -(kd + c\dot{d})n^{P_{ij}} = f_n n^{P_{ij}} \tag{8}$$

where k and c are spring and damping coefficients, respectively, and f_n is the magnitude of the normal force component.

The tangential friction forces can be evaluated using the LuGre friction model [31] which accounts for both static and sliding phenomena based on a bristle deflection interpretation. Accordingly, the LuGre model captures the dynamic behavior of the contact surface using the first order differential equation for bristle deflections, which can be written in vector form as follows:

$$\dot{z} = \dot{s}_t - \sigma_0 \frac{\mid \dot{s}_t \mid}{g(\dot{s}_t)} z \tag{9}$$

where z is bristle deflection and σ_0 is the stiffness coefficient of the contacting surfaces. In (9), $g(\dot{s}_t)$ is used to capture the Stribeck effect [32] in order to describe stick-slip phenomena, and can be calculated as follows:

$$g(\dot{s}_t) = \alpha_0 + \alpha_1 ^{-(\frac{\dot{s}_t^T \dot{s}_t}{\dot{s}_0^2})} \tag{10}$$

where x_0 is the Stribeck velocity and the parameters α_0 and α_1 are defined as follows:

$$\alpha_0 = F_n \mu_d \tag{11}$$

$$\alpha_1 = F_n(\mu_s - \mu_d) \tag{12}$$

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

$$F_t = \sigma_0 z + \sigma_1 \dot{z} + c\dot{s}_t \tag{13}$$

where σ_1 is the friction damping coefficient. For bodies i and j, the resulting contact force can be applied as follows:

$$F_C^i = F_C \tag{14}$$

$$F_C^j = -F_C \tag{15}$$

Accordingly, the resulting torque of contact can be written as follows:

$$T_{F_C}^i = \widetilde{A^i \bar{u}^{P_i}} F_C^i \tag{16}$$

$$T_{F_C}^j = \widetilde{A^j \bar{u}^{P_j}} F_C^j \tag{17}$$

where $\widetilde{A^i \bar{u}^{P_i}}$ and $\widetilde{A^j \bar{u}^{P_j}}$ are skew-symmetric matrices.

Having these equations, the forces and torques can be calculated on each contact point. These forces are applied to the body where the tactile sensor is attached as well as to the body that the tactile sensor is colliding with. They are also used to retrieve sensor feedback information.

In summary, these are the steps needed to construct the simulated tactile sensor and to calculate the contact forces:

- Create a mesh representing the real sensor geometry. 2b).
- Create the simulated sensor elements by placing a vertex on the center of each of the mesh polygons.
- Parametrize the sensor elements with the maximum penetration and the values needed to calculate friction (k, c, μ_s, μ_d, σ_0, σ_1, α_1).
- Place a vector pointing to each vertex of the simulated sensor element with a magnitude equal to the maximum penetration.
- Each time-step, calculate the intersection of this vector with the target objects. If they are in collision, create a contact point on this intersection.
- on each contact point, calculate the contact force in the normal direction (8) and the tangential friction (13), and add these two components to get the contact force.
- Calculate and apply the forces (14,13) and torques (16, 17) to the sensor and target bodies.
- Convert the forces to tactile values.
- Compare this tactile values with the real tactile sensor values and adjust the conversion if necessary.

IV. Implementation on the OpenGRASP Toolkit

The tactile sensor model presented in the previous section has been implemented using OpenRAVE [33], a planning architecture developed at the Carnegie Mellon University Robotics Institute. It is an open architecture targeting a simple integration of simulation, visualization, planning, scripting and control of robot systems. It enables the user to easily extend its functionality developing its own custom plugins.

Following its design, the Tactile Sensor Plugin has been developed and is available in OpenGRASP [34], a simulation toolkit for grasping and dexterous manipulation consisting of a set of OpenRAVE plugins and other tools like the RobotEditor.

The development of the tactile sensor plugin included the definition of the tactile sensor geometry, the tactile sensor data and the implementation of the sensor interface specified by OpenRAVE.

The tactile sensor requires a set of parameters to be specified for each sensor. The model presented on this study allows the creation of sensors based on any geometry which is specified by the mesh of the body to which the sensor is attached. For each sensor, the parameters necessary to calculate the contact forces (see Section III) and its thickness need to be defined. An example of a sensor definition using the OpenRAVE XML format can be seen in Table I.

TABLE I
EXAMPLE OF AN OPENRAVE TACTILE SENSOR DEFINITION

```
< AttachedSensor >
< link > object1 < /link >
< sensorname = "T1"  type = "SimTactileSensor" >
  < shapeBaseOnGeom > true < /shapeBaseOnGeom >
  < thickness > 0.005 < /thickness >
  < sigma0[a]> 2e2 < /sigma0 >
  < sigma1[b]> 5e − 1 < /sigma1 >
  < alfa1[c]> 10.5 < /alfa1 >
  < mus[d]> 0.9 < /mus >
  < mud[e]> 0.6 < /mud >
  < k > 1e3 < /k >
  < c > 1e1 < /c >
< /sensor >
< /AttachedSensor >
```

[a] sigma0=σ_0 [b] sigma1=σ_1 [c] alfa1=α_1 [d] mus=μ_s [e] mud=μ_d

The tactile sensor data is the structure returned by the tactile sensor when it is requested. It contains the size of the tactile array, a vector with the tactile values calculated in each cell and the sum of all the tactile values. This structure is the same as the one used by the real tactile sensor, which makes the real and simulated sensors feedback appear identical to the controllers.

The plugin is an implementation of the sensor interface. OpenRAVE creates a new tactile sensor when specified on a robot definition. Each time step, when the sensor is updated, it gets the positions and velocities of the sensor and objects and checks if they are colliding. On each contact point, it calculates the contact forces with the equations detailed in Section III-C. These forces and torques are applied to the sensor and objects. Finally, the tactile intensities are calculated using the linear conversion and the tactile data structure is updated. Controllers can query and use this tactile sensor feedback as required.

V. EXPERIMENTS ON ROBOT GRASPING

In order to determine the simulated tactile sensor performance, some experiments were carried out.

The basic physical properties of the simulated sensor were tested in various ways. To verify the theoretical correctness of the equations used in the force calculation a certain load was set on top of the sensor and the tactile force sum was compared to the load applied. Also the friction properties of the tactile sensor were tested by moving an object in contact with the sensor at an increasing velocity. The friction force data was then plotted to determine the shape of the friction force in relation to the sliding velocity. The simulated tactile sensor passed these tests successfully.

However, the focus of this section is to present the results when testing the sensor performance in a real grasping situation. The natural selection of a test case was a simple grasping task. The selected test case was executed by the real robot after which the same situation was replicated using the simulator. The model was then modified to validate the consistency of the simulation results.

A. Experiment Setup

A simple task of grasping and picking up a cube using a Schunk PG70 parallel jaw gripper was selected as the test scenario. Each finger of the gripper had a DSA 9205 tactile sensor attached to it. The tactile sensor feedback was used to control the grasping force and to determine the stability of the grasp.

The idea was to perform the same task using this robot and compare the results with the ones obtained by executing the same actions in the simulator. In order to accomplish this, a high level controller was implemented using a new abstraction architecture presented in [35] which is also available in OpenGRASP [36]. This architecture uses a hierarchical approach for decomposing a manipulation task into a sequence of actions represented as finite state machines. Each action consists of primitive actions implemented using a single low-level controller responsible for the control of the robot hardware. This approach ensures the ability to use it with different embodiments and to use multiple sensors and sensor types.

Following its design, the experiment was defined as an abstract action consisting of five primitive actions: approach, grasp, lift, move down and release. The controller turns this abstract information into the gripper specific primitives and transitions. It then drives the Schunk actuator using velocity control until the first contact with the tactile sensors is detected. After the initial touch the controller switches to force control by setting the maximum current of the Schunk gripper based on the tactile sensor feedback. The tactile value sums are used as the reference for the force control. This particular case shows the function of the tactile sensor in combination with the robot controller.

Given the abstraction architecture's ability to be embodiment independent, the same controller was used to control the real robot as well as the simulated case. This feature makes possible to demonstrate the capabilities of the simulated tactile sensor in comparison to the real one.

B. Experimental Results for the Real Robot

The real robot performed the task close to what was expected. An image sequence of the real work cycle with the tactile images produced by each sensor, can be seen in Fig. 4. The experiment was performed with the tactile sensor covering 30 percent of its area when touching the cube.

The actions were executed as expected but closer investigation of the tactile values revealed some problems.

When grasping the cube, even pressure was applied to each tactile sensor that should return, as a result, very similar tactile images. However, significant differences between the individual tactile elements can be seen in the bottom row of Fig. 4 where one sensor tactile image is substantially lighter than the other in most cases, when touching the cube.

The tactile value sum retrieved for the tactile sensors in each time-step can be seen in Fig. 5a. This graph also shows that opposite tactile sensors give out significantly different value sums when they should have been very similar. It can also be seen that the controller has difficulties in reaching the desired tactile value which is most likely due to the friction in the

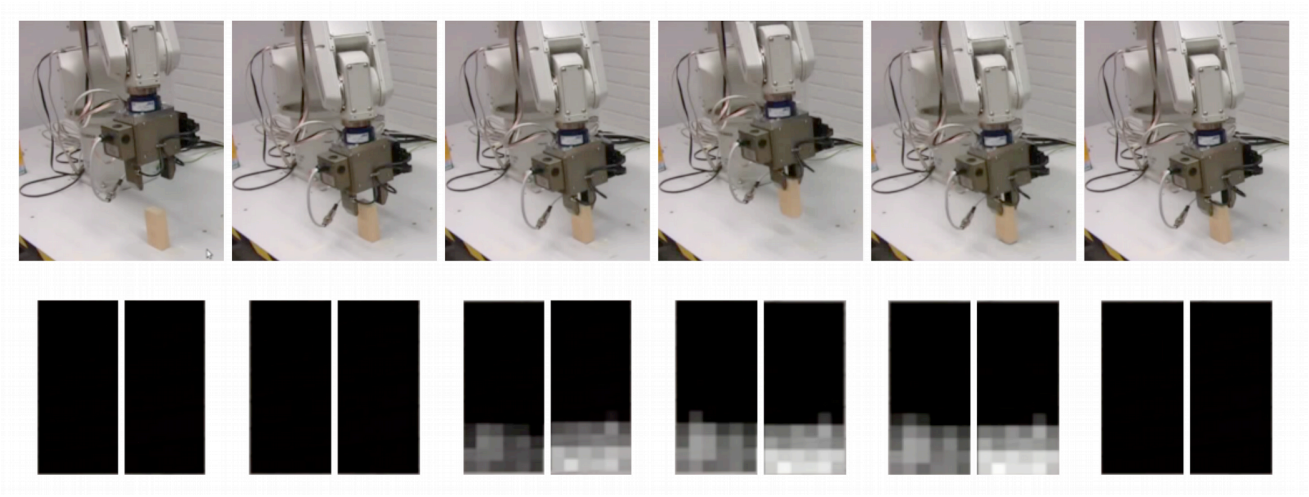
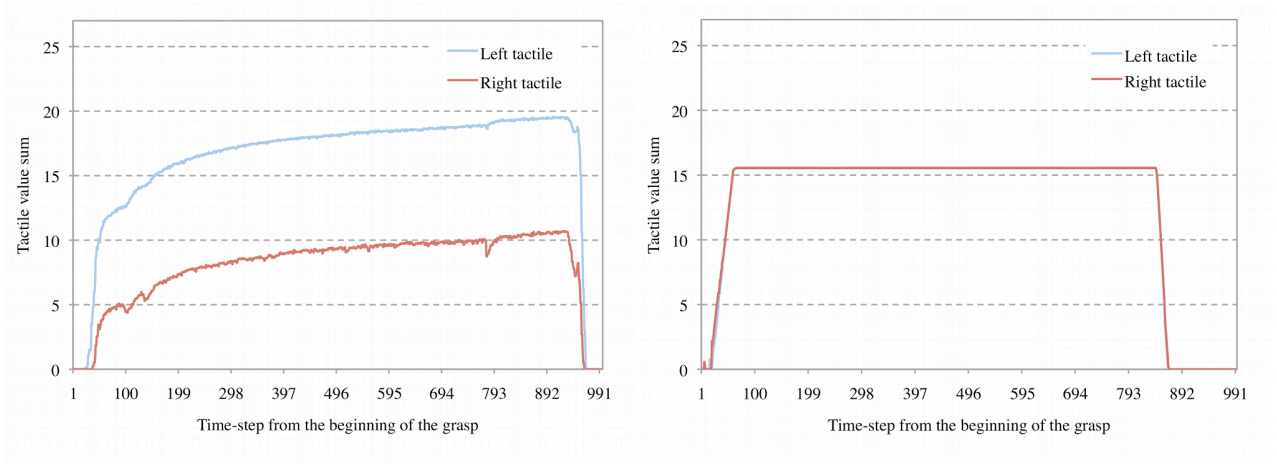


Fig. 4. Real robot performing the chosen work cycle. On the top row, pictures of the robot performing the task on each stage. On the bottom, the tactile images generated by left and right sensors.



(a) (b)

Fig. 5. Tactile value sums retrieved by the tactile sensors each time-step when grasping a cube covering 30% of the sensor tactile area: (a) real robot and (b) simulated robot

Schunk gripper. The low velocity stick-slip friction can cause the controller to not reach this value.

C. Experimental Results for the Simulated robot

The simulated robot performed the same work cycle as the real one. Images of the robot on each work cycle phase are shown in Fig. 6 with the corresponding tactile images at the bottom.

Closer inspection of the tactile values reveals that the sensors perform exactly as expected. Under even pressure, the individual tactile element values are the same in the area touching the cube. Opposite sensors also return similar tactile values. The tactile value sum retrieved for the simulated tactile sensors in each time-step can be seen in Fig. 5b. This graph also shows that opposite tactile sensors give out very similar values and that the tactile value sum rises to the desired value swiftly.

To test the performance of the tactile sensors under different conditions the same work cycle was performed on the simulator by changing the robot's starting height. This resulted in the gripper grasping the cube using different coverage percentages of the tactile sensor. The results are shown in Fig. 7 and reveal the consistency in all trials of this experiment. The tactile model becomes stiffer the more surface is touching the object. This can be easily seen in the rising slopes of the diagrams. The softest case with 15 percent coverage rises to the desired value much more slowly than the stiffest case at 100 percent coverage. The change in the controller stage can also be seen quite clearly in the beginning of the graph where there is a jump in the value when the controller switches from velocity to force control. The stiffer the system the uneasier the transition to a stable grasp is.

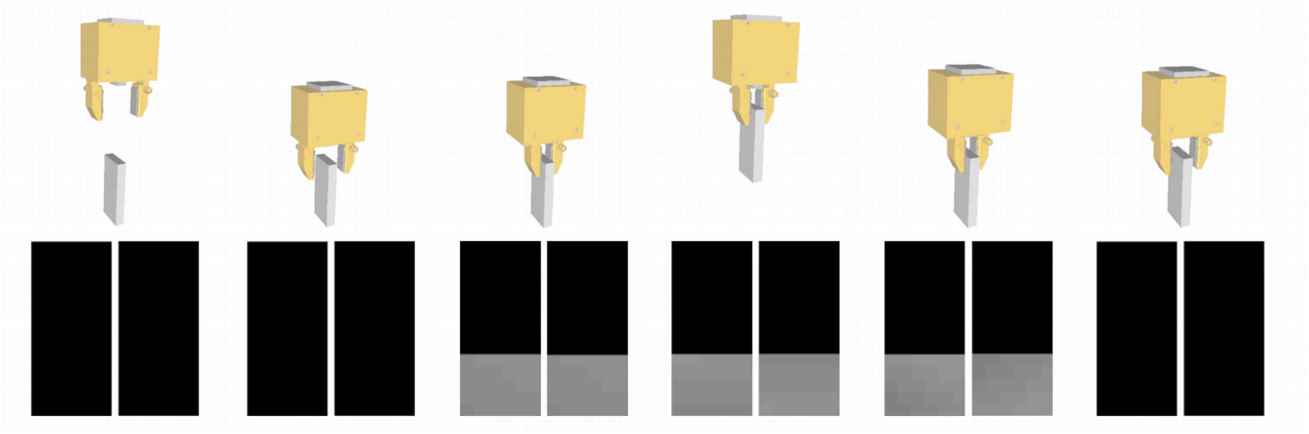
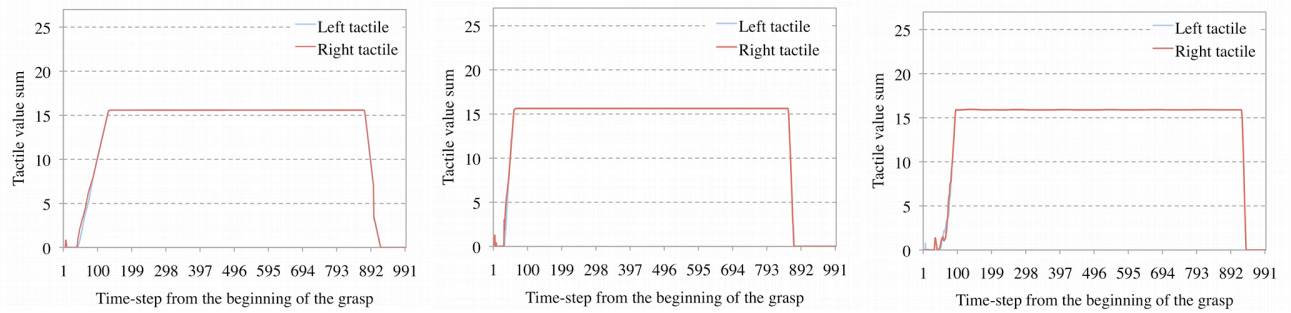


Fig. 6. Simulated robot performing the chosen work cycle. On the top row, pictures of the simulated robot performing the task on each stage. On the bottom, the tactile images generated by left and right sensors.



(a) (b) (c)

Fig. 7. Tactile value sums from the different test executed by the simulated robot varying the robot's starting height, and thus covering: (a) 15%, (b) 60% and (c) 100% of the tactile sensor area.

D. Comparing Experimental Results

The simulation trials show that the simulated tactile sensor element can be used to perform as the real one. The simulation model shows consistent results whereas the real tactile sensor results vary on each work cycle. This is due to the fact that in simulation there are no manufacturing flaws or problems from wearing. The system consistently performs the same way under the same conditions. In addition, the detection tolerances from individual elements do not pose a problem.

If the tactile images in the real (Fig. 4) and simulated trials (Fig. 6) are compared, it can be seen that the tactile values differ. These differences are explained by the non-ideal performance of the real tactile elements. It can also be seen that the shape of the tactile value sum curves (Fig. 5a) and (Fig. 5b) also differ. This can be due to the missing calculation of the stiction in the simulated gripper actuator. When using low velocities and forces, stiction becomes a significant factor in force control. In this respect the simulator behaves ideally allowing the controller to reach the desired value without the accuracy problem caused by the stick-slip phenomenon.

Using the most important features of a tactile sensor [10] (spatial and temporal resolution, noise, hysteresis, creep and aging) as criteria, the simulated and real sensor can also be compared. The real sensor's spatial and temporal resolution are both hardware dependent. In the simulated sensor the resolution can be changed somewhat freely. Currently the simulated sensor stiffness changes when adding resolution which causes the need for changing the model parameters. From the real sensor's results it can also be seen that the tactile values suffer from noise in the results whereas the simulated sensor reports the changes in the force directly without any interference or error caused by the hardware. The hysteresis in the real tactile can also be significant due to the material covering the tactile elements. The hysteresis effect is included in the simulation model as a damping value which can be controlled. The real sensor material also causes some creep in the results as the foam cover resistance changes over time even under constant pressure. This change settles after some time but there is always some creep even after an extended period. The simulated sensor does not suffer from this creep as there is no material modelled. The cover material also causes aging to be a problem when using the real tactile.

All these features of the real tactile sensor can be added to the tactile sensor model. The difficulty is that the variance from sensor to sensor can be quite considerable as shown in (Fig. 5a).

E. Shape Recognition Using the Simulated Robot

The simulated robot was further tested by running a scenario where the simulated tactile sensor was used to recognize object shapes given that, as mention in the introduction, this is one of the main applications of this type of sensor.

In these experiments, the target object was modified to have differently-shaped holes on each side. One side of the cube has a cross-shaped hole and the other a square, as shown in Fig. 8. This image shows the tactile values from each sensor and its respective position on each of the graspable faces of the cube. As it can be seen, the tactile sensors give out consistent results. The pressures in the opposite sides are not exactly even as the graspable cube was not centered between the fingers at the beginning of the grasp. This enables the cube to stick to the table thus creating an uneven pressure spread on the sensor.

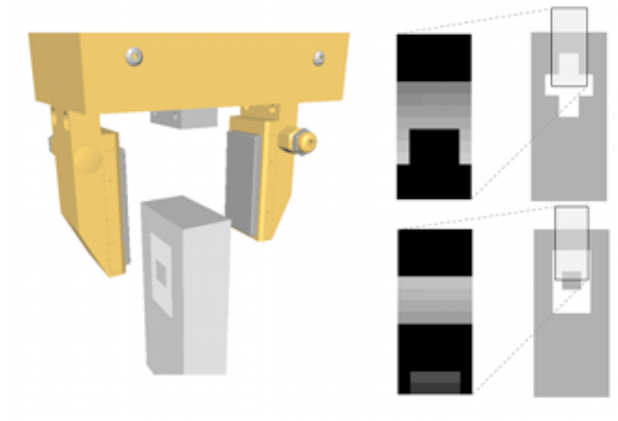


Fig. 8. Experiment showing tactile sensor values that can be used for shape recognition. On the left, the simulated robot grasping the cube. On the right, tactile images and positions of the left and right simulated tactile sensors showing the shape of the face they are touching.

VI. Conclusion

In this study, a simulation model of a tactile sensor was presented. The simulation model is based on soft contact modelling with a full friction description. The sensor was tested using the most common use cases of tactile sensors in robotic grasping. The simulated tactile sensor performed all the tests including stable grasping as well as pattern recognition without any errors and can be used as an ideal universal tactile sensor model. The model can be updated to behave in the exact same manner as a specified type of tactile sensor such as one from Weiss Robotics. This would entail modifying the stiffness to be non-linear as well as adding delays that are due to the covering material and other electrical properties.

The experiment results of the tactile sensor model show good performance in being able to produce tactile feedback. Problems arise when trying to calibrate the tactile model to correspond exactly to a certain real tactile sensor. This is due to the variations in the real tactile values which makes the process extremely difficult.

Due to the tolerances in the real sensors, the simulated sensors function better than the real ones. This enables researchers to do experiments that should be theoretically possible but, due to the current limitations in the existing hardware, are still difficult.

The experiments also show that in order to accurately model the whole control system, the actuator models need to be upgraded to include better friction models. This would allow the simulator to be used in non-ideal conditions.

The tactile model can also be made more accurate. Currently, it is a perfectly linear measuring unit. In the Weiss Robotics sensor the covering material greatly determines the properties of the measurements. This material effect can be added by measuring the compression of different materials and adjusting the sensor model's normal force accordingly. The material seems to be adding a slight delay in the sensor, which could also be taken into account. These modifications would however be Weiss-sensor specific and not in accordance with a universal tactile sensor model.

The model was not made to be used in real time simulations and therefore is not optimized in terms of calculation speed. Currently it uses a brute force method for the collision search algorithm, checking all possible primitives. This makes the simulations slow. The work cycle for the real robot takes about 45 seconds where as the simulation takes approximately 2 minutes on an Intel Pentium 2.8 GHz dual core processor. The simulation efficiency can be greatly improved by adding optimizations to this algorithm.

References

[1] R. Dahiya, G. Metta, M. Valle, and G. Sandini, "Tactile sensing - from humans to humanoids," *Robotics, IEEE Transactions on*, vol. 26, no. 1, pp. 1 –20, feb. 2010.

[2] J. Tegin and J. Wikander, "Tactile sensing in intelligent robotic manipulation a review," *Industrial Robot: An International Journal*, vol. 32, no. 1, pp. 64–70, 2005.

[3] J. Felip and A. Morales, "Robust sensor-based grasp primitive for a three-finger robot hand," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 10-15 2009, pp. 1811 – 1816.

[4] D. Kragic, S. Crinier, D. Brunn, and H. Christensen, "Vision and tactile sensing for real world tasks," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 1, 14-19 2003, pp. 1545 – 1550 vol.1.

[5] M. Prats, P. Martinet, S. Lee, and P. Sanz, "Compliant physical interaction based on external vision-force control and tactile-force combination," in *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, 20-22 2008, pp. 405 –410.

[6] M. Prats, P. J. Sanz, and A. P. Pobil, "Reliable non-prehensile door opening through the combination of vision, tactile and force feedback," *Auton. Robots*, vol. 29, no. 2, pp. 201–218, 2010.

[7] K. Hsiao, S. Chitta, M. Ciocarlie, and E. G. Jones, "Contact-reactive grasping of objects with partial shape information," in *Intelligent Robots and Systems, 2010. IROS 2010. IEEE/RSJ International Conference on*, October 2010, to be published.

[8] Y. Bekiroglu, J. Laaksonen, J. Jrgensen, V. Kyrki, and D. Kragic, "Learning grasp stability based on haptic data," in *In Proceedings of the RSS 2010 Workshop: Representations for object grasping and manipulation in single and dual arm tasks*, June 2010.

[9] Y. Hasegawa, M. Shikida, T. Shimizu, T. Miyaji, H. Sasaki, K. Sato, and K. Itoigawa, "micromachined active tactile sensor for hardness detection," *Sensors and Actuators A: Physical*, vol. 114, no. 2-3, pp. 141 – 146, 2004, selected papers from Transducers 03.

[10] A. Schneider, J. Sturm, C. Stachniss, M. Reisert, H. Burkhardt, and W. Burgard, "Object identification with tactile sensors using bag-of-features," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 10-15 2009, pp. 243 –248.

[11] G. Heidemann and M. Schöpfer, "Dynamic tactile sensing for object identification," in *Proc. IEEE Int. Conf. Robotics and Automation ICRA 2004*, IEEE. New Orleans, USA: IEEE, 2004, pp. 813–818.
[12] A. M. Okamura and M. R. Cutkosky, "Feature Detection for Haptic Exploration with Robotic Fingers," *The International Journal of Robotics Research*, vol. 20, no. 12, pp. 925–938, 2001.
[13] 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.
[14] 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.
[15] D. Baraff, "Analytical methods for dynamic simulation of non-penetrating rigid bodies," *SIGGRAPH Comput. Graph.*, vol. 23, no. 3, pp. 223–232, 1989.
[16] B. Mirtich, "Impulse-based dynamic simulation of rigid body systems," Ph.D. dissertation, University of California, Berkeley, 1996.
[17] 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.
[18] 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.
[19] K. Hunt and F. Crossley, "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.
[20] Y. Khulief and A. Shabana, "A continuous force model for the impact analysis of flexible multibody systems," *Mechanism and Machine Theory*, vol. 22, no. 3, pp. 213 – 224, 1987.
[21] K. Johnson, "Contact mechanics," in *Cambridge University Press*. University Press, 1985, p. 452.
[22] G. Hippmann, "An algorithm for compliant contact between complexly shaped bodies," *Multibody System Dynamics*, vol. 12, no. 4, pp. 345–362, December 2004.
[23] 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.
[24] ODE, open dynamics engine. [Online]. Available: http://www.ode.org/
[25] Bullet, game physics simulation. [Online]. Available: http://www.bulletphysics.org/
[26] Weiss Robotics DSA 9205 Tactile Transducer. [Online]. Available: http://www.weiss-robotics.de/en/products/tactile-transducers/99-dsa-9205.html
[27] 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.
[28] D. Goger and H. Worn, "A highly versatile and robust tactile sensing system," in *Sensors, 2007 IEEE*, 28-31 2007, pp. 1056 –1059.
[29] D. Goger, N. Gorges, and H. Worn, "Tactile sensing for an anthropomorphic robotic hand: Hardware and signal processing," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, 12-17 2009, pp. 895 –901.
[30] D. Kubus, R. Iser, S. Winkelbach, and F. M. Wahl, "Efficient parallel random sample matching for pose estimation, localization, and related problems," in *Advances in Robotics Research*, T. Krger and F. M. Wahl, Eds. Springer Berlin Heidelberg, 2009, pp. 239–250.
[31] 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.
[32] B. Armstrong-Helouvry, P. Dupont, and C. de Wit., "A survey of models, analysis tools and compensation methods for the control of machines with friction," *Automatica*, vol. 30, no. 7, pp. 1083–1138, Mar 1994.
[33] 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.
[34] B. León, S. Ulbrich, R. Diankov, G. Puche, M. Przybylski, A. Morales, T. Asfour, S. Moisio, J. Bohg, J. Kuffner, and R. Dillmann, "OpenGRASP: A toolkit for robot grasping simulation," in *SIMPAR '10: Proceedings of the 2st International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2010, to be published.
[35] J. Laaksonen, J. Felip, A. Morales, and V. Kyrki, "Embodiment independent manipulation through action abstraction," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 3-7 2010, pp. 2113 –2118.
[36] OpenGRASP, a simulation toolkit for grasping and dexterous manipulation. [Online]. Available: http://opengrasp.sourceforge.net

Sami Moisio was born in Lappeenranta Finland, in 1977. He received the M.Sc. degree in Mechanical Engineering from Lappeenranta University of Technology, Finland in 2005.

During 2005-2010 he has been working on several projects financed by the Academy of Finland and the National Technology Agency of Finland (TEKES), latest being EU-funded project GRASP. He is currently working as a Ph. D. student in Lappeenranta University of Technology, Finland. His research interest covers topics related to mechatronics and virtual engineering especially real-time simulation software development and collision modelling.

Beatriz León (M'09) was born in Bogotá, Colombia in 1979. She received the B.Sc. in Computer Science from the University Antonio Nariño, Bogotá, Colombia, the B.Sc. in Civil Engineering from the National University of Colombia and the M.Sc. in Autonomous Systems from the Bonn-Rhein-Sieg University of Applied Sciences (Förderpreis 2008), Bonn, Germany in 2001, 2003, and 2008 respectively.

From 2006 to 2008, she was part of the EU-funded project XPERO. Since September 2008, she has been with the Robotic Intelligence Lab, Universitat Jaume I, Castellón, Spain where she is involved in the EU-funded project GRASP. She is currently a Ph. D. student in this university. Her current research interests include areas of application of simulation to robotic grasping and the development of a virtual bio-mechanical model of the human hand.

Pasi Korkealaakso was born in Finland, in 1977. He received the M.Sc. degree in Mechanical Engineering from Lappeenqranta University of Technology, Finland, in 2002, and the Ph.D. degree from Lappeenranta University of Technology, Finland, in 2009.

During 2002-2010 he has been part of several research projects financed by the Academy of Finland and the National Technology Agency of Finland (TEKES), latest being EU-funded project GRASP. He is currently a Post-Doctoral Researcher in Lappeenranta University of Technology, Finland. His research interest covers topics related to mechatronics and virtual engineering especially to development of multibody simulation algorithms and real-time simulation software development. Dr. Korkealaakso is Co-owner and Technical director at MeVEA Ltd.

Antonio Morales (M'00) was born in Castellón, Spain, in 1973. He received a B.Sc in Computer Engineering and a M.Sc. degree in Advanced Informatics Systems from Universitat Jaume I, Castellón, Spain, in 1996 and 1999 respectively. He completed the Ph.D on 2004 from Universitat Jaume I, Castellón, Spain.

In 1999, he stayed as guest researcher at the Institute for Autonomous Systems at the GMD - National German Research Center for Information Technology, in Bonn, Germany. In 2000-04 and 06-07 he was teaching assistant at University Jaume I. During 2005 he served as post-doc researcher at Institute of Computer Science and Engineering (IAIM), in the University of Karlsruhe, Germany collaborating in the German Humanoid Robot Research Project (SFB588). Since 2008, he is holds a position of Associate Professor at the Department of Computer Science and Engineering at Universitat Jaume I, in Castellón. His research interest focus on robot grasping and manipulation and robot simulation.

Mind the Gap - Robotic Grasping under Incomplete Observation

Jeannette Bohg, Matthew Johnson-Roberson, Beatriz León, Javier Felip,
Xavi Gratal, Niklas Bergström, Danica Kragic and Antonio Morales

***Abstract*— We consider the problem of grasp and manipulation planning when the state of the world is only partially observable. Specifically, we address the task of picking up unknown objects from a table top. The proposed approach to *object shape prediction* aims at closing the knowledge gaps in the robot's understanding of the world. A completed state estimate of the environment can then be provided to a simulator in which stable grasps and collision-free movements are planned.**

The proposed approach is based on the observation that many objects commonly in use in a service robotic scenario possess symmetries. We search for the optimal parameters of these symmetries given visibility constraints. Once found, the point cloud is completed and a surface mesh reconstructed.

Quantitative experiments show that the predictions are valid approximations of the real object shape. By demonstrating the approach on two very different robotic platforms its generality is emphasized.

I. Introduction

Many challenging problems addressed by the robotics community are currently studied in simulation. Examples are motion planning [1], [2], [3] or grasp planning [4], [5], [6] in which the knowledge of the complete world model or of specific objects is assumed to be known. However, on a real robotic platform this assumption breaks down due to noisy sensors or occlusions. Consider for example the point cloud in Fig. 1 that was collected with an active stereo head from a table top scene with several objects standing on it.

Due to occlusions, no information about the backside of objects or about the region behind them is available from pure stereo reconstructed data. There are *gaps* in the scene and object representations. Additionally, noise causes the observed 3D structure of an object or scene to deviate from its true shape. Although the previously mentioned planners might be applied on real robotic platforms, a mismatch between the real world and the world model is usually not dealt with explicitly.

Exceptions are reactive grasping strategies that adapt their behavior based on sensor information collected during execution time [7], [8]. Other approaches aim at predicting unknown parts of the world and plan the robot's course of action based on this [9].

In this paper, we consider the scenario of picking up unknown objects from a table top. Therefore, the robot is confronted with scenes similar to those in Fig. 1. To accomplish grasp and motion planning based on this information, we follow the idea of filling in the gaps in the scene representation through predicting the full shape of each object. We make use of the observation that many, especially man-made objects, possess one or more symmetries. Given this, we can provide the simulator with an estimated complete world model under which it can plan actions or predict sensor measurements.

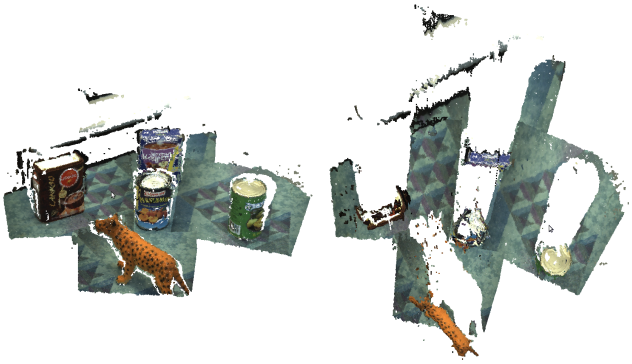


Fig. 1. Example for a point cloud representing a whole scene merged from different view points. Left: View from the side. Right: View from the top.

The contributions of this paper are (i) a quantitative evaluation of the shape prediction on real-world data containing a set of objects observed from a number of different viewpoints. This is different from related work by Thrun and Wegbreit [10] in which only qualitative results in form of point clouds are presented rather than quantitative results on polygonal meshes. (ii) We reduce the search space for the optimal symmetry parameters through a good initialization. And (iii), we demonstrate the applicability of the predicted object meshes in a service robotic scenario by supporting execution in the real-world with grasp and motion planning in simulation.

The paper is outlined as follows. In the next section, we will motivate the use of the symmetry assumption and explain the method for object shape prediction and polygonal mesh reconstruction. Thereafter, the two experimental platforms are described. In Section IV the simulating environment OpenGRASP is presented. In the experimental section, we show how the proposed prediction mechanism produces valid complete object models and is advantageous for grasp planning and execution.

II. Predicting Object Shape through Symmetry

Estimating the occluded and unknown part of an object has applications in many fields, e.g. 3D shape acquisition,

This work was supported by the EU through the project GRASP, IST-FP7-IP-215821. J. Bohg, M. Johnson-Roberson, X. Gratal, N. Bergström and D. Kragic are with CVAP/ CAS at KTH, Stockholm, Sweden. {bohg, mattjr, gratal, nbergst, dani}@csc.kth.se.
B. León, J. Felip and A. Morales are with the Robotic Intelligence Laboratory at the Department of Computer Science and Engineering, UJI, Castellón, Spain {len, jfelip, morales}@uji.es

3D object recognition or classification. In this paper, we are looking at this problem from the perspective of service robotics and are therefore interested in the advantages of shape completion for collision detection and grasp planning.

Psychological studies suggest that humans are able to predict the portions of a scene that are not visible to them through *controlled scene continuation* [11]. The expected structure of unobserved object parts are governed by two classes of knowledge: i) Visual evidence and ii) completion rules gained through prior visual experience. A very strong prior that exists in especially man-made objects is symmetry.

In [10] it has been shown that this symmetry can be detected in partial point clouds and then exploited for shape completion. The authors developed a taxonomy of symmetries in which *planar reflection symmetry* is the most general one. It is defined as the case in which each surface point P can be uniquely associated with a second surface point Q by reflection on the opposite side of a symmetry plane. Furthermore, in a household environment, objects are commonly placed such that one of their symmetry planes is perpendicular to the supporting plane. Exceptions exist such as grocery bags, dishwashers or drawers.

Given these observations, for our scenario we can make the assumption that objects commonly possess one or several planar symmetries of which one is usually positioned perpendicular to the table from which we are grasping. By making these simplifications, we can reduce the search space for the pose of this symmetry plane significantly. As it will be shown in the experimental section, our method produces valid approximations of the true object shape in very different viewpoints and for varying levels of symmetry.

A. Detecting Planar Symmetry

Since we assume the symmetry plane to be perpendicular to the table plane, the search for its pose is reduced to a search for a line in the 2 1/2D projection of the partial object point cloud. This line has 3 degrees of freedom (DoF): the 2D position of its center and its orientation. We follow a generate-and-test scheme in which we create a number of hypotheses for these three parameters and determine the *plausibility* of the resulting mirrored point cloud based on visibility constraints.

We bootstrap the parameter search by initializing it with the major or minor eigenvector e_a or e_b of the projected point cloud. As shown in Section V, this usually yields a good first approximation. Further symmetry plane hypotheses are generated from this starting point by varying the orientation and position of the eigenvectors as outlined in Fig. 2. In the following, we will describe the details of this search.

1) Initial Plane Hypothesis: The first hypothesis for the symmetry axis is either one of the two eigenvectors of the projected point cloud. Our goal is to predict of the unseen object part. We therefore make the choice dependent on the inverse viewing direction v: the eigenvector that is most perpendicular to v is used as the symmetry plane s.

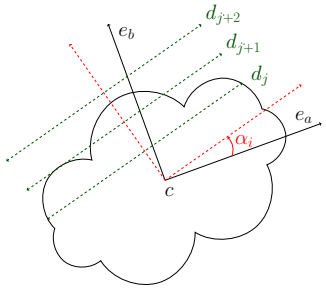


Fig. 2. A set of hypotheses for the position and orientation of the symmetry plane. e_a and e_b denote the eigenvectors of the projected point cloud. c is its center of mass. α_i denotes one of the variations of line orientation along which the best pose of the symmetry plane is searched. The (green) lines at positions d_j to d_{j+2} are three further candidates with orientation α_i.

$$s = \begin{cases} e_a \text{ if } e_a \cdot v \leq e_b \cdot v \\ e_b \text{ if } e_a \cdot v > e_b \cdot v \end{cases} \tag{1}$$

where e_a and e_b denote the major and minor eigenvectors of the projected point cloud, respectively.

2) Generating a Set of Symmetry Hypothesis: Given this initial approximation of the symmetry plane s of the considered point cloud, we sample n line orientations α_i in the range between $-20°$ and $20°$ relative to the orientation of s. The 2D position of these n symmetry planes is varied based on a shift d_j in m discrete steps along the normal of the symmetry plane. Together this yields a set $\mathcal{S} = \{s_{(0,0)} \cdots s_{(0,j)} \cdots s_{(i,j)} \cdots s_{(n,m)}\}$ of $n \times m$ hypotheses.

3) Mirroring the Point Cloud: Let us denote the original point cloud as $\mathcal{P}$ containing points P. Then given some symmetry plane parameters $s_{(i,j)}$, the mirrored point cloud $\mathcal{Q}_{(i,j)}$ with points Q is determined as follows:

$$Q = R_{\alpha_i}^{-1}(-R_{\alpha_i}(P + d_j - c)) + c \tag{2}$$

with R_{α_i} denoting the rotation matrix corresponding to α_i.

4) Computing the Visibility Score: The simplest source of information about visibility constraints are the binary 2D segmentation masks $\mathcal{O}$ that separate an object from the background. The mirroring process aims at adding points Q to the scene that were unseen from the original viewpoints.

Let us assume a mirrored point cloud $\mathcal{Q}_{(i,j)}$ has been generated, then there are three cases for where a reflected point Q can be positioned. i) If it coincides with a point in $\mathcal{P}$ (the original point cloud), then it supports the corresponding symmetry hypothesis. ii) If Q falls into previously occluded space, it provides information about potential surfaces not visible from the original viewpoint. And finally iii), if Q is positioned into the space that has been visible before, then it contradicts the symmetry hypothesis.

Based on this intuition, the vote $v_{(i,j)}$ consists of two parts. First, we back project each $\mathcal{Q}_{(i,j)}$ into the original image giving us a set of pixels. For all these pixels q that do not lie inside the original object segmentation mask $\mathcal{O}$, we compute the squared distance $\delta_1(q,p)$ to the nearest pixel p in the segmentation mask using the distance transform. Second, for all q that lie within the segmentation mask and have a

smaller depth relative to the camera than the corresponding (occluded) pixel p, we compute the depth difference $\delta_2(q,p)$ between them. Summarizing, the vote $v_{(i,j)}$ is computed as the sum of the expected values of these two distance measures:

$$v_{(i,j)} = \mathop{\mathrm{E}}_{q \notin \mathcal{O}}[\delta_1(q,p)] + \mathop{\mathrm{E}}_{q \in \mathcal{O}}[\delta_2(q,p)]. \quad (3)$$

In case the plane parameters are chosen such that there is a large overlap between the original point cloud $\mathcal{P}$ and the mirrored cloud $\mathcal{Q}_{(i,j)}$, the second part of Eq. 3 will be relatively large compared to the first part. The bigger d_j, i.e. shift of the symmetry plane as visualized in Fig. 2, $\mathrm{E}_{q \in \mathcal{O}}[\delta_2(q,p)]$ will increase and $\mathrm{E}_{q \notin \mathcal{O}}[\delta_1(q,p)]$ decrease. We are searching for the global minimum in the space of all votes

$$\hat{s}_{(i,j)} = \arg\min_{S} v_{(i,j)} \quad (4)$$

that corresponds to a reflected point cloud $\mathcal{Q}_{(i,j)}$ with the smallest amount of points that contradict the symmetry hypothesis.

B. Surface Approximation

After the prediction of the backside of an object point cloud, we create a surface mesh approximation to support grasp planning and collision detection.

We use Poisson reconstruction proposed by Kazhdan et. al [12] as a solution to the problem of surface reconstruction from oriented points. To determine the normals, we use a kd-tree as proposed in [13]. A local plane fit is estimated for the k-nearest neighbors of the target point. This plane is assumed to be a local approximation of the surface at the current point. More advanced normal estimation techniques have been proposed which could perhaps increase the performance of the surface reconstruction at the cost of computation speed [14], [15]. Following normal estimation, we ensure that the normals of the mirrored points are consistent with the mirrored viewing direction reflected across the plane of symmetry.

Finally the Poisson reconstruction is performed. The single steps are briefly outlined below. For details, we refer to [12].

1) Inputting the points and normals to an octree.
2) Computing an implicit function over this adaptive grid.
3) Finally using marching cubes to extract an iso-surface as a watertight triangular mesh.

It should be noted while Poisson surface reconstruction is robust to some noise, it is sensitive to normal direction. We therefore filter outliers from the segmented point cloud prior to the reconstruction step.

III. Experimental Platforms and Grasp Cycle

To emphasize the generality of the proposed approach, we evaluate and demonstrate it on two robotic platforms being either used at the Royal Institute of Technology (KTH) or at the Universitat Jaume I (UJI). They differ in both, hardware and implementation of a grasp cycle.

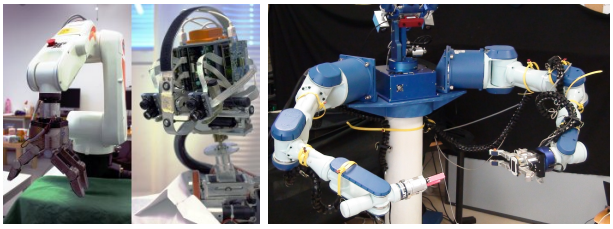


Fig. 3. Left: KTH robot. Right: UJI robot (Tombatossals)

A. Hardware

1) KTH: The platform (see Fig. 3 left) consists of an Armar III robotic head [16] equipped with two stereo cameras, a peripheral (wide-field) and a foveal (narrow-field) one. The robotic head has 7 DoF. Five of these are used for controlling the viewing direction while the remaining two mainly vary the vergence angle between the left and right camera systems, thereby enabling fixation on objects. As a manipulator, we use a 6 DoF Kuka arm[1] that is equipped with a three-fingered Schunk Dexterous Hand 2.0 (SDH)[2].

2) UJI: The torso system, called *Tombatossals* has 23 DOF (see Fig. 3 right). It is composed of two 7 DOF Mitsubishi PA10 arms. The left arm has a 4 DOF Barrett Hand[3] and the right arm has a parallel jaw gripper. Each arm has a JR3 Force-Torque sensor attached on the wrist between the arm and the hand. The visual system is composed of a TO40 4 DOF pan-tilt-verge head with two Imaging Source DFK 31BF03-Z2 cameras. Attached to the center of the pan-tilt there is a Videre DCSG-STOC stereo camera. For this work only the left arm, the pan-tilt head and the Videre stereo system are used.

B. Grasp Cycle

For the scenario of grasping unknown objects from a table top, a grasp cycle is outlined in Algorithm 1. The functions serve as place holders for operations that are common to both robotic platforms but are implemented differently. Exceptions to this are the function `PredictObjectShape`, the simplification of the triangular mesh through `ConvexDecomposition` and the grasp and arm trajectory planners named `PlanGrasp` and `PlanArmTrajectory`. They are identical in both systems and are explained in Section II and Section IV, respectively. The remaining functions of Algorithm 1 are not the focus of this paper. Therefore, in the below we will only briefly outline how they are implemented in each robotic system and refer to our previous work.

1) KTH: As a step prior to the shape analysis of an object and the grasping of it, the robot needs to segregate potential objects from the background. In the KTH system, the function `GetObjectHypotheses` to obtain a segmented point cloud is implemented as explained in detail in our previous work [17]. Given this point cloud which only represents the visible part of an object, its backside can be predicted as described in Section II.

[1] `http://www.kuka-robotics.com`
[2] `http://www.schunk.com`
[3] `http://www.barrett.com`

Algorithm 1: Pseudo Code for a Table Top Scenario

```
Data: Embodiment, Table Plane
Result: Cleaned Table Top
begin
    S = GetObjectHypotheses()
    for i = 0; i < |S|; i++ do Grasp Cycle per Object Hypothesis
        ŝ_i = PredictObjectShape(s_i)
        ŝ_i = ConvexDecomposition(s_i)
        G = GetGraspCandidates(ŝ_i)
        for j = 0; j < |G|; j++ do
            success = PlanGrasp(g_j)
            if success then Grasp Candidate gives Stable Grasp
                t_(i,j) = PlanArmTrajectory(g_j)
                if t_(i,j) ≠ 0 then Collision-Free Trajectory Exists
                    T = InsertTrajectory(T, t_(i,j), g_j)
                end
            end
        end
        k = 0
        repeat
            success = ExecuteGrasp(T, k)
            k++
        until success
    end
end
```

With the complete object shape as an input, `GetGraspCandidates` implements the technique presented in [18] to detect two grasping points. These points are the target positions for the fingers of the SDH. In detail, we apply a pinch grasp in which the thumb is opposite the two fingers. Then for applying a grasp to an object, the vector between the thumb and the two fingers has to be aligned with the vector between the two grasping points.

These grasp candidates are then simulated on the predicted object shape and a collision-free path for the arm planned. Details on this will be given in Section IV. After a suitable grasp and arm trajectory has been selected through simulation, it is executed by the robot in an open loop procedure.

2) UJI: The vision system on the UJI platform implementing `GetObjectHypotheses` is quite simple. The Videre stereo system gathers images and produces an unsegmented 3D point cloud of the scene. The table and background are black to simplify the segmentation. The point cloud is segmented finding its connected components using a clustering method as implemented in ROS PCL [4].

The complete object shape is predicted as described in Section II. `GetGraspCandidates` is implemented such that the vector between the thumb and two fingers is going through the object's centroid (see Section IV for more detail).

Approaching the pre-grasp position is also executed in open loop following the collision free trajectory obtained from the simulator. For the grasping action, a reactive sensor based strategy is used. This algorithm is fully described in [7], basically it tries to align the hand with the object and performs a power grasp adapting the hand pose to the object pose using tactile and force feedback.

IV. OpenGRASP

The simulation platform chosen to perform the experiments is OpenGRASP [19], a simulation toolkit for grasping and dexterous manipulation. It is based on the OpenRAVE [20], an open architecture targeting a simple integration of simulation, visualization, planning, scripting and control of robot systems. It enables the user to easily extend its functionality developing their own custom plugins.

The simulator is used to perform the grasp before the real robot makes an attempt. It allows for testing different alternatives, choosing the one with the highest probability of success. This will not only take considerably less time than performing it with the real hardware but also prevents damaging the robot by avoiding collisions with the environment.

OpenRAVE implements a combined motion and grasp planner plugin [2]. This BiSpace planner has elements from the bidirectional RRT and the RRT-JT algorithms [3]. The use of RRTs [1] is a well known approach to the arm motion planning problem. It has been the starting point for most of the current state of the art motion planners. For collision detection that is necessary for motion planning, the simulator needs complete object models. When known objects are used, an accurate model of the objects can be created off-line using different technologies, like laser scans. In the case of unknown objects, an approximate model has to be created on-line. In Section II, an approach to create this model was presented and it is evaluated in Section V. The experimental results show that even if the object model is not an exact representation of reality, it is close enough to enable the simulator to try different grasp alternatives and select an appropriate one.

The obtained triangular mesh has thousands of vertices which makes the collision detection process computationally expensive. To ameliorate this problem, we pre-process the mesh by `ConvexDecomposition`, a library that was originally created by John Ratcliff [21] and that is implemented in OpenRAVE. It approximates a triangular mesh as a collection of convex components. This process takes only a few seconds and drastically speeds up the grasp and motion planning.

A. Generation of Grasp Candidates

The first step for selecting an appropriate grasp consists of creating a set of grasp candidates and evaluating them using OpenRAVE. This set can be stored and used later, anytime the same robot has to grasp the same object. Each grasp candidate is simulated moving the end-effector until it collides with the object; then the fingers will close around it and finally the contacts are used to test on force closure. In Algorithm 1, this process is referred to as `PlanGrasp`.

Each grasp candidate has the following parameters: the approach vector, the hand pre-shape, the approach distance and the end-effector roll. The number of different values that these parameters can take has to be chosen considering the time-constraints imposed by the on-line execution of `PlanGrasp`.

OpenRAVE has a default algorithm to generate a set of approach vectors. It first creates a bounding box of the object and samples its surface uniformly. For each sample, a ray is intersected with the object. At each intersection, an approach

[4] http://www.ros.org/wiki/pcl

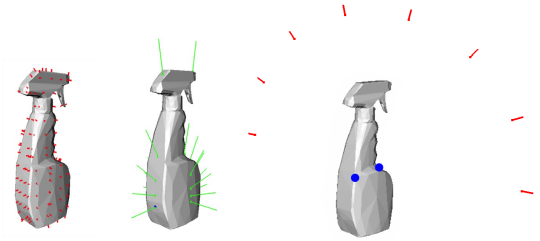


Fig. 4. Example of the approach vectors generated for a spray bottle by Left) the OpenRAVE grasper plugin, Middle) the UJI proposed algorithm using the object's centroid and Right) the KTH proposed algorithm using the grasp points in blue.

vector is created that is aligned with the normal of the object's surface at this point. An example output is shown in Fig. 4 Left). Dependent on the choice of the other parameters, the time to simulate all the corresponding grasps can vary from few minutes to more than an hour. These execution times are acceptable for objects that are known beforehand because the set of grasp candidates can be generated off-line. When the objects are unknown, this process has to be executed on-line and long waiting times are not desirable.

For this reason, we use two methods to reduce the number of approach vectors. The first one, applied on the KTH platform, computes two grasp points as in [18]. To grasp the object at these points, there are an infinite number of approach vectors on a circle with the vector between the two grasping points as its normal. We sample a given number, typically between 5 and 10 of these between 0° and 180° degrees. Figure 4 (Right) shows the detected grasping points along with the generated approach vectors. The second method, used at UJI, calculates the approach vectors in a similar way only that the center of the circle is aligned with the object's centroid and its major eigenvector e_a. Another circle, perpendicular to the first one, is added in order to compensate the possible loss of vector quality due to the lack of grasp points. Figure 4 (Middle) shows an example.

Having the list of approach vectors reduced, the other parameters were also adjusted for our purposes. As a hand pre-shape, we defined a pinch grasp for each hand. The approach distance is varied between 0 to 20 cm. Finally, the roll is chosen dependent on the two grasping points (such that the fingers are aligned with them) or on the orientation of the circle on which the selected approach vector is defined. Using these parameters, we were able to reduce the amount of time taken to generate and save the set of grasp candidates to less than a minute.

B. Grasp Execution Using Motion Planners

The next step `PlanArmTrajectory` in Algorithm 1, consists of selecting a stable grasp from the set of grasp candidates that can be executed with the current robot configuration without colliding with obstacles. For each stable grasp, it first moves the robot to the appropriate grasp pre-shape, then uses RRT and Jacobian-based gradient descent methods to move the hand close to the target object, closes the fingers, grabs the object, moves it to the destination while avoiding obstacles and releases it.

If the robot successfully grabs the object and moves it to the destination, the stable grasp is returned for execution with the real robot. Otherwise, the next stable grasp from the set is tried. Fig. 5 shows snapshots of the grasp execution using the simulator and the real robots.

V. Experiments

In this section, we will present quantitative experiments showing that the completion of incomplete object point clouds based on symmetry produces valid object models. Furthermore, we will show how the estimated complete object model helps when using a simple grasp strategy based on the center of an object.

A. Evaluation of the Mesh Reconstruction

In this section, we will evaluate how much the reconstructed mesh differs from the ground truth mesh.

1) Dataset: The database we used for evaluating the point cloud mirroring method is shown in Figure 6. For each of the objects in this database, with the exception of the toy tiger and rubber duck, we have laser scan ground truth[5]. The test data was captured with the KTH vision system and contains 12 different household or toy objects. Four of them are used both, when standing upright or lying on their side. Thereby, the database contains 16 different data sets. Each set contains 8 stereo images showing the object in one of the following orientations: 0°, 45°, 90°, 135°, 180°, 225°, 270° or 335°. An example for the toy tiger is shown in Figure 7. As an orientation reference we used the longest object dimension when projected down to the table. 0° then means that this reference axis is parallel to the image plane of the stereo camera. This can be observed in Figure 6 in which all objects are shown in their 0° pose. Therefore, the database contains 128 stereo images along with their point clouds.

From all point clouds, we reconstructed the complete meshes based on the method described in Section II. We used the two different values of 5 and 7 as the octree depth parameter of the Poisson surface reconstruction. By limiting this parameter, we enable mesh reconstruction in near real-time. With a tree depth of 5, the meshes are more coarse and blob-like but less sensitive to noise in the point cloud and normal estimation. With a depth of 7, the reconstructed surface is closer to the original point set. However, outliers strongly affect the mesh shape and it is more sensitive to noise.

To obtain the ground truth pose for each item in the database, we applied the technique proposed in [22]. It allows to register the laser scan object meshes to the incomplete point clouds.

2) Baseline: As a baseline, we reconstructed a mesh without mirroring. To do this, we applied a Delaunay triangulation[6] to the projection of a uniformly sampled subset of 500 points from the original point cloud. Spurious edges are filtered based on their length in 2D and 3D. Furthermore,

[5]The ground truth object models were obtained from `http://i61p109.ira.uka.de/ObjectModelsWebUI/`

[6]http://opencv.willowgarage.com

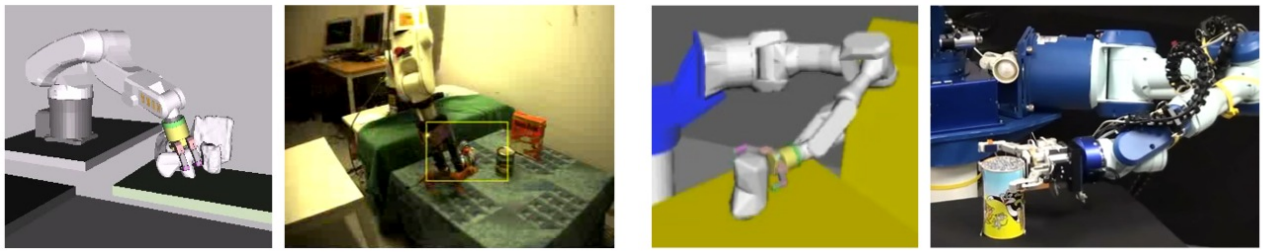


Fig. 5. Example of the grasp performed by the simulated robot and the real one, using Right) KTH platform and Left) UJI platform.

Fig. 7. One of the Datasets from Figure 6 shown in Orientation 0°, 45°, 90°, 135°, 180°, 225°, 270° and 335°.

Fig. 6. The 12 Objects in the Database in varying poses yielding 16 Data Sets. Objects are shown in their 0° position, i.e, with their longest dimension parallel to the image plane. Ground truth meshes are existing for all objects except the toy tiger and the rubber duck.

we extract the outer contour edges of this triangulation and span triangles between them which produces a watertight mesh. Figure 8 shows the result of this Delaunay based mesh reconstruction for the toy tiger.

3) Mesh Deviation Metric: To assess the deviation of the Delaunay meshes and the mirrored meshes from the ground truth we use MeshDev [23]. As a metric we evaluated geometric deviation, i.e., the distance between each point on the reference mesh to the nearest neighbor on the other mesh. We applied the uniform sampling of the surface of the reference mesh as proposed in [23] to calculate this deviation.

4) Results: Figure 10 shows the mean and variance of the mesh deviation between the ground truth mesh and the reconstructed meshes for all object orientations over all objects. We can state that the mirrored point clouds are on average always deviating less from the ground truth than

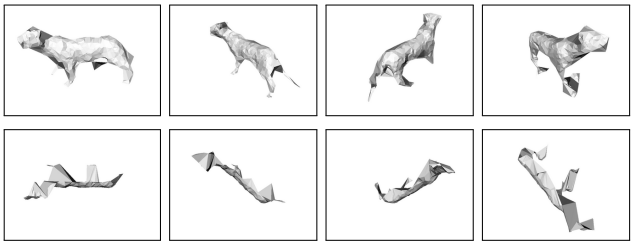


Fig. 8. Delaunay based Meshes of Toy Tiger in the following Orientations: 0°, 45°, 135°, 225°. 1-4: Front View. 5-8: Top View.

the Delaunay based meshes. The average deviation for the mirrored meshes over all orientations amounts to $7mm$.

Figure 11 shows the same error measure for each object independently averaged over all its orientations. The deviation measure is not normalized to the overall object size. Therefore, for bigger objects, like the *Brandt* box or the *Burti* and *Spray* bottle, the mean geometric deviation between the Delaunay meshes and the ground truth exceed 20mm. The mirroring yields a significant improvement for most objects. The green and white cup pose a challenging problem to the Poisson surface reconstruction because they are hollow. Modelling the void is difficult due to viewpoint limitations. On the other hand, holes in the point clouds due to non-uniform texture are usually closed by the surface reconstruction.

Since, we do not have the ground truth models available for the toy tiger and the rubber duck, we show their reconstructed meshes with tree depth 7 in Figure 9. The overall shape of the quite irregular toy objects is well reconstructed. However, because of the complexity of the objects if an incorrect mirroring plane is chosen, we obtain toy animals with either two heads or two tails. In such cases, there are strong violations of the visibility constraints. Thresholding of the vote in Equation 3 could address this. This is considered as future work.

B. Deviation of the Object Centroid

A very simple but effective grasping strategy of unknown objects is to approach the object at its center. However, estimating the centroid is not a trivial problem when the

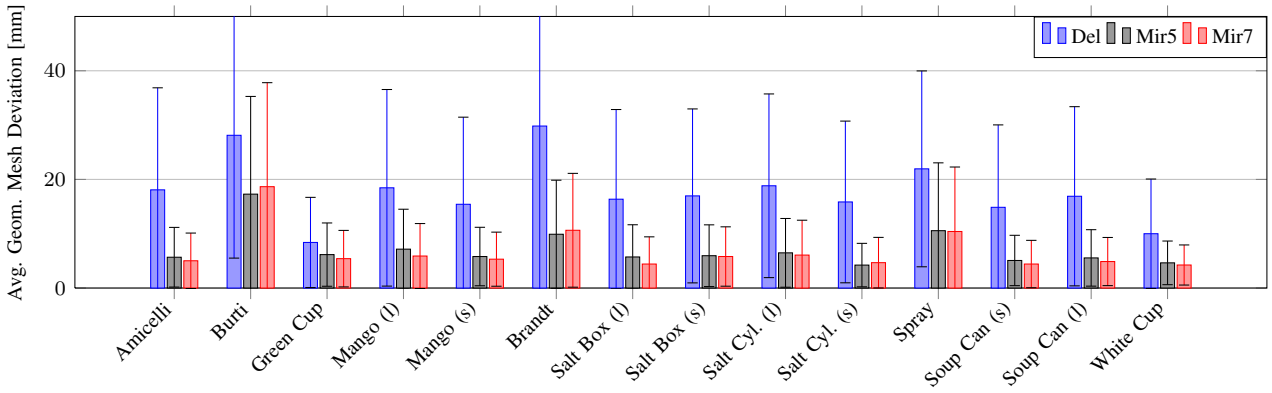


Fig. 11. Evaluation of the Deviation between the Ground Truth Mesh and i) Mesh based on 3D Point Cloud only (*Del*), ii) Mesh based on mirroring and Poisson Surface Reconstruction with Tree Depth 5 (*Mir5*) or iii) with Tree Depth 7 (*Mir7*). Pose of mirroring plane chosen over a set of different positions and orientations. Mean and Variance are computed for each object over all eight orientations.

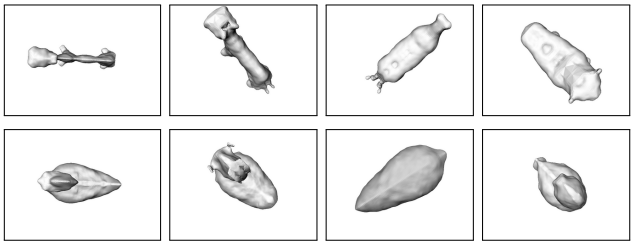


Fig. 9. Meshes based on Mirroring. First Row: Toy Tiger. Second Row: Rubber Duck.

object is unknown. Reactive grasping strategies are proposed to cope with the uncertainty in the object information during the grasp [8], [7]. The method proposed in [7] is applied for the grasp execution on the UJI platform. The more accurate the initial estimate of the object centroid, the fewer unnecessary contacts with the object occur in a reactive grasping scheme.

In this section, we therefore evaluated the accuracy of object center estimation as a simple placeholder for grasp quality. We compared the center of mass of the Delaunay mesh and of the mirrored mesh with the ground truth center. To render this comparison independent of the distribution of vertices (especially for the ground truth meshes), we applied the same uniform sampling of the mesh surface as in the previous section [23]. The center of mass is then the average over all the samples.

Figure 12 shows the error between the estimated and real centroid of an object per viewing direction and averaged over all objects. The deviation is normalized with the length of the diagonal of the oriented object bounding box. We can observe that the deviation ranges from approximately 5% to 10% of the total object size.

C. Real-World Experiments

We demonstrated the approach proposed in this paper on the two robotic platforms described in Section III. A video of the experiment can be found at `http://opengrasp.sourceforge.net/Videos/BohgICRA11.mp4`. Fig. 5 shows snapshots of the grasp execution in simulation on the predicted objects and with the real robots on the

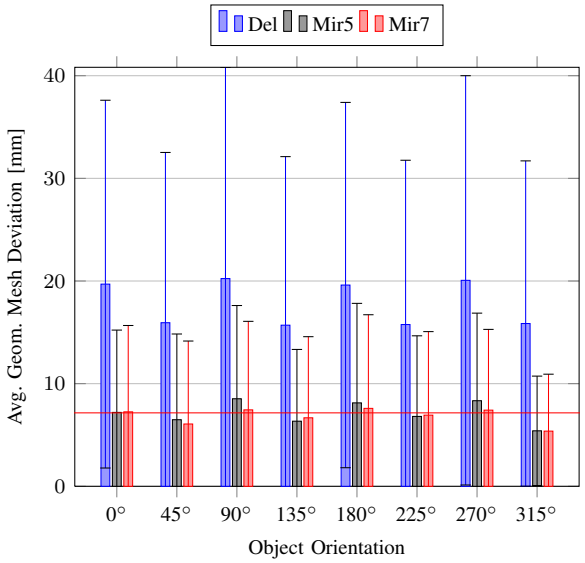


Fig. 10. Evaluation of the Deviation between the Ground Truth Mesh and i) Mesh based on 3D Point Cloud only (*Del*), ii) Mesh based on mirroring and Poisson Surface Reconstruction with Tree Depth 5 (*Mir5*) or iii) with Tree Depth 7 (*Mir7*).

real objects. At KTH, several objects were placed on the table emphasizing the benefit of object shape prediction for motion planning. Furthermore, it shows that the prediction mechanism can deal with some occlusions. This is due to the enforced visibility constraints. One of the main differences between the runs at UJI and KTH is the resolution of the point clouds that is due to the use of camera systems with different focal lengths. While the KTH point clouds usually consist of 40000 points, UJI point clouds contained around 3000 points. However, a suitable mesh could still be generated with the advantage of a lower runtime. When running the whole generate and test procedure (with $n = 6$ and $m = 5$ yielding 35 hypotheses) on a single core of an I7 CPU with 2.8 GHz, we achieved the following run-times: 16.46 seconds for a point cloud with 39416 points and 0.31 seconds for a point cloud with 2100 points. Please note, that we have not exploited the possibility to parallelize this

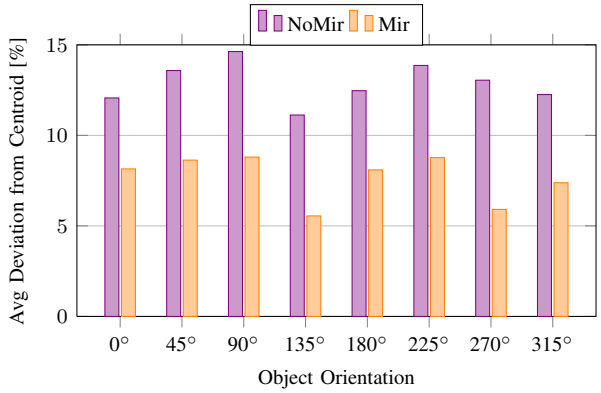


Fig. 12. Deviation of Estimated Object Centroid from Ground Truth Centroid.

process yet. These runtimes also show that downsampling the point clouds before mirroring can speed up the shape completion without a big loss of precision. To investigate these optimizations is considered as future work.

VI. Conclusions

In this paper, a method that estimates complete object models from partial views is proposed. We have validated these complete models using laser scan ground truth. The results show effectiveness of the technique on a variety of household objects in table top environments. Furthermore, the proposed technique has been demonstrated on two different robotic platforms, validating the feasibility of the predicted mesh for grasp and motion planning.

We feel that the proposed technique is a first step towards bridging the gap between simulation and the real world. In future, we hope to develop planners that take uncertain shape information explicitly into account to generate better grasp hypotheses and motion plans.

References

[1] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *ICRA'2000: Proceedings of the 2000 IEEE/RSJ international conference on Robotics and Automation*. Piscataway, NJ, USA: IEEE Press, 2000.

[2] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner, "Bispace planning: Concurrent multi-space exploration," in *Robotics: Science and Systems*, June 2008.

[3] M. Vande Weghe, D. Ferguson, and S. Srinivasa, "Randomized path planning for redundant manipulators without inverse kinematics," in *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, nov. 2007, pp. 477 –482.

[4] M. Ciorcarlie, C. Goldfeder, and P. Allen, "Dexterous Grasping via Eigengrasps: A Low-Dimensional Approach to a High-Complexity Problem," *Robotics: Science and Systems Manipulation Workshop*, 2007.

[5] C. Goldfeder, P. K. Allen, C. Lackner, and R. Pelossof, "Grasp Planning Via Decomposition Trees," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 4679–4684.

[6] A. T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen, "Automatic Grasp Planning Using Shape Primitives," in *IEEE Int. Conf. on Robotics and Automation*, 2003, pp. 1824–1829.

[7] J. Felip and A. Morales, "Robust sensor-based grasp primitive for a three-finger robot hand," in *IROS'09: Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1811–1816.

[8] K. Hsiao, S. Chitta, M. Ciocarlie, and E. G. Jones, "Contact-reactive grasping of objects with partial shape information," in *IROS'10: Proceedings of the 2010 IEEE/RSJ international conference on Intelligent robots and systems*, October 2010.

[9] J. Bohg, M. Johnson-Roberson, M. Björkmann, and D. Kragic, "Strategies for multi-modal scene exploration," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Taipei,Taiwan, October 2010.

[10] S. Thrun and B. Wegbreit, "Shape from symmetry," *Computer Vision, IEEE International Conference on*, vol. 2, pp. 1824–1831, 2005.

[11] T. P. Breckon and R. B. Fisher, "Amodal volume completion: 3d visual completion," *Comput. Vis. Image Underst.*, vol. 99, no. 3, pp. 499–526, 2005.

[12] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006, pp. 61–70.

[13] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *COMPUTER GRAPHICS-NEW YORK-ASSOCIATION FOR COMPUTING MACHINERY-*, vol. 26, pp. 71–71, 1992.

[14] D. Cole, A. Harrison, and P. Newman, "Using naturally salient regions for SLAM with 3D laser data," in *International Conference on Robotics and Automation, SLAM Workshop*. Citeseer, 2005.

[15] N. Mitra and A. Nguyen, "Estimating surface normals in noisy point cloud data," in *Proceedings of the nineteenth annual symposium on Computational geometry*. ACM, 2003, p. 328.

[16] T. Asfour, K. Welke, P. Azad, A. Ude, and R. Dillmann, "The Karlsruhe Humanoid Head," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, Daejeon, Korea, December 2008.

[17] X. Gratal, J. Bohg, M. Björkman, and D. Kragic, "Scene representation and object grasping using active vision," in *IROS 2010 Workshop: Defining and Solving Realistic Perception Problems in Personal Robotics*, Taipei, Taiwan, October 2010.

[18] M. Richtsfeld and M. Vincze, "Grasping of Unknown Objects from a Table Top," in *ECCV Workshop on 'Vision in Action: Efficient strategies for cognitive agents in complex environments'*, Marseille, France, September 2008.

[19] B. León, S. Ulbrich, R. Diankov, G. Puche, M. Przybylski, A. Morales, T. Asfour, S. Moisio, J. Bohg, J. Kuffner, and R. Dillmann, "OpenGRASP: A toolkit for robot grasping simulation," in *SIMPAR '10: Proceedings of the 2st International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2010.

[20] 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.

[21] J. Ratcliff. (2009) Convex decomposition library. [Online]. Available: http://codesuppository.blogspot.com/2009/11/convex-decomposition-library-now.html

[22] C. Papazov and D. Burschka, "Stochastic optimization for rigid point set registration," in *ISVC '09: Proceedings of the 5th International Symposium on Advances in Visual Computing*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 1043–1054.

[23] M. Roy, S. Foufou, and F. Truchetet, "Mesh Comparison Using Attribute Deviation Metric," *International Journal of Image and Graphics*, vol. 4, 2004.