frontiers in Robotics and AI

CODE
published: 15 November 2017
doi: 10.3389/frobt.2017.00059

Real-time Pipeline for Object Modeling and Grasping Pose Selection *via* Superquadric Functions

Giulia Vezzani[1,2]* *and Lorenzo Natale*[1]

[1] iCub Facility, Istituto Italiano di Tecnologia, Genova, Italy, [2] University of Genova, Genova, Italy

This work provides a novel real-time pipeline for modeling and grasping of unknown objects with a humanoid robot. Such a problem is of great interest for the robotic community, since conventional approaches fail when the shape, dimension, or pose of the objects are missing. Our approach reconstructs in real-time a model for the object under consideration and represents the robot hand both with proper and mathematically usable models, i.e., *superquadric functions*. The volume graspable by the hand is represented by an ellipsoid and is defined *a priori*, because the shape of the hand is known in advance. The superquadric representing the object is obtained in real-time from partial vision information instead, e.g., one stereo view of the object under consideration, and provides an approximated 3D full model. The optimization problem we formulate for the grasping pose computation is solved online by using the Ipopt software package and, thus, does not require off-line computation or learning. Even though our approach is for a generic humanoid robot, we developed a complete software architecture for executing this approach on the iCub humanoid robot. Together with that, we also provide a tutorial on how to use this framework. We believe that our work, together with the available code, is of a strong utility for the iCub community for three main reasons: object modeling and grasping are relevant problems for the robotic community, our code can be easily applied on every iCub, and the modular structure of our framework easily allows extensions and communications with external code.

Keywords: grasping, object modeling, real-time optimization, C++, superquadric functions

OPEN ACCESS

Edited by:
Maxime Petit,
Imperial College London,
United Kingdom

Reviewed by:
Tobias Fischer,
Imperial College London,
United Kingdom
Uriel Martinez-Hernandez,
University of Leeds,
United Kingdom
Ingo Keller,
Heriot-Watt University,
United Kingdom

***Correspondence:**
Giulia Vezzani
giulia.vezzani@iit.it

Specialty section:
This article was submitted to Humanoid Robotics, a section of the journal Frontiers in Robotics and AI

Received: *28 July 2017*
Accepted: *30 October 2017*
Published: *15 November 2017*

Citation:
Vezzani G and Natale L (2017) Real-time Pipeline for Object Modeling and Grasping Pose Selection via Superquadric Functions. Front. Robot. AI 4:59. doi: 10.3389/frobt.2017.00059

1. INTRODUCTION

Industrial robotics shows how high performance in manipulation can be achieved if a very accurate knowledge of the environment and the objects is provided. On the contrary, grasping of unknown objects or whose pose is uncertain is still an open problem. In this work, we present a novel framework for modeling and grasping unknown objects with the iCub humanoid robot.

The iCub humanoid robot is provided with two 7DOF arms, 5 fingers human-like hands, whose fingertips are covered by tactile sensors and two cameras, as described in Metta et al. (2010). Therefore, it turns out to be a suitable platform for investigating objects perception and grasping problem: the stereo vision system and the tactile sensors can be exploited together to get proper information for modeling and grasping unknown objects. The method and the code, we propose in this work, consist of reconstructing an object model through the stereo vision system of the robot and using this information to compute a suitable grasping pose. Once the robot reaches the desired grasping pose on the object surface, the tactile response of the fingertips is used to achieve a stable grasp for lifting the object.

The iCub community put a great effort into the development of a sharable and reusable code. With this work, we want to contribute in this direction, detailing the code we designed for implementing our grasping approach for a possible user interested in executing our technique on the robot.

2. MODELING AND GRASPING *VIA* SUPERQUADRIC MODELS

The superquadric modeling and grasping framework we make use of is based on the idea that low-dimensional, compact, mathematical representation of objects can provide computational and theoretical advantages in hard problems tackled in robotics, such as trajectory planning for exploration, grasping and approaching toward objects. This takes inspiration from theories conceived during the 90s and 2000s (Jaklic et al., 2013) where superquadric functions were proposed as a mathematical and low-dimensional model for representing objects.

In Vezzani et al. (2017), we proposed a novel approach that solves the grasping problem by modeling the object and the volume graspable by the hand with superquadric functions. The latter is represented by an ellipsoid and is defined *a priori*, because the shape of the hand is known in advance. The superquadric representing the object is obtained in real-time from partial vision information instead, e.g., one stereo view of the object under consideration, and provides an approximated 3D full model. Both the modeling and the grasping problem are cast into an optimization framework and solved in real-time with the software package Ipopt (Wächter and Biegler, 2006).

In this article, we do not go into the mathematical details (extensively reported in Vezzani et al. (2017)) whereas we focus on the description of the code designed for using the approach on the iCub, since we believe it to be useful for any user interested in object modeling and grasping tasks. A brief mathematical description of the methodologies is reported in the *README.md* files of the Github repositories.[1]

3. CODE STRUCTURE

We designed two modules, namely, *superquadric-model* and *superquadric-grasp*, which implement, respectively, the modeling and the grasping approached described in Vezzani et al. (2017).

Our leading idea is to develop a *self-contained* code that provides *query services* to the user. In this respect, our code handles only the information strictly necessary for the superquadric modeling and grasping approach and minimizes the dependencies from external modules. The user is asked to write a wrapper code that communicates with the two modules and makes them properly interact. In this respect, we provide a tutorial code,[2] implementing a possible use case of our modules, that can be adapted by the user to fit in his own pipeline (see Section 3.3).

In the next paragraphs, we first describe the implementation of the *superquadric-model* and *superquadric-grasp* modules, which is based on the *Yarp* middleware (Metta et al., 2006). Then, we outline a possible use case implementing a complete pipeline for object modeling and grasping.

3.1. Superquadric-Model

The *superquadric-model* module computes the superquadric function best representing the object of interest given a partial 3D point cloud of the object.

The module, whose structure is outlined in **Figure 1**, consists of the *SuperqModule* class, derived from the YARP *RFModule* class. The SuperqModule launches following two separate YARP *Rate Threads*:

- the *SuperqComputation* class, which manages the superquadric computation;
- the *SuperqVisualization* class, which can be enabled to show the estimated superquadric or the object 3D points overlapped on the camera image.

The SuperqModule also provides some *Thrift IDL services*[3] suitable for getting information on the internal state of the module and setting the thread parameters on the fly. *Thrift* is a software framework for scalable cross-language development, which allows to build services working efficiently with different programming languages.

While there are two threads to decouple the functionalities of computation and visualization, the threads share some variables (in particular the computed superquadric) to increase their speed.

3.1.1. SuperqComputation

The *SuperqComputation* thread includes the following steps:

- Once the object point cloud is provided (see Section 3.3 for a detailed description of how extract the object point cloud), the superquadric is estimated by using Ipopt (Wächter and Biegler, 2006), a C++ software package for large-scale nonlinear optimization. The user can formulate its own optimization problem with the Ipopt C++ interface[4] and, then, solve it through the Ipopt solver.
- A median filter with an adaptive window of width m can be enabled to stabilize the estimated superquadric over the time. Even if the object is not supposed to move during a grasping task, it may happen that the user, or anyone interacting with the robot, moves the object in a different location. In this case, the superquadric modeler should be able to track the object and the estimated superquadric should not be affected by previous estimations in different poses. For this reason, the window width of the median filter changes according to the object velocity. If the object location changes (i.e., its velocity increases), the window width becomes smaller. On the contrary, if the object is not moved, the window width can be increased. In this way, when the object pose is constant, its superquadric estimation is more stable and accurate, while it is not affected by past estimations if the object pose changes. The median filter and the object velocity estimation are achieved by using, respectively, the *iCub MedianFilter Class* and the *iCub AWLinEstimator Class*.

[1] https://github.com/robotology/superquadric-model,https://github.com/robotology/superquadric-grasp.

[2] https://github.com/robotology/superquadric-grasp-example.

[3] https://thrift.apache.org/docs/idl.

[4] https://www.coin-or.org/Ipopt/documentation/node23.html.

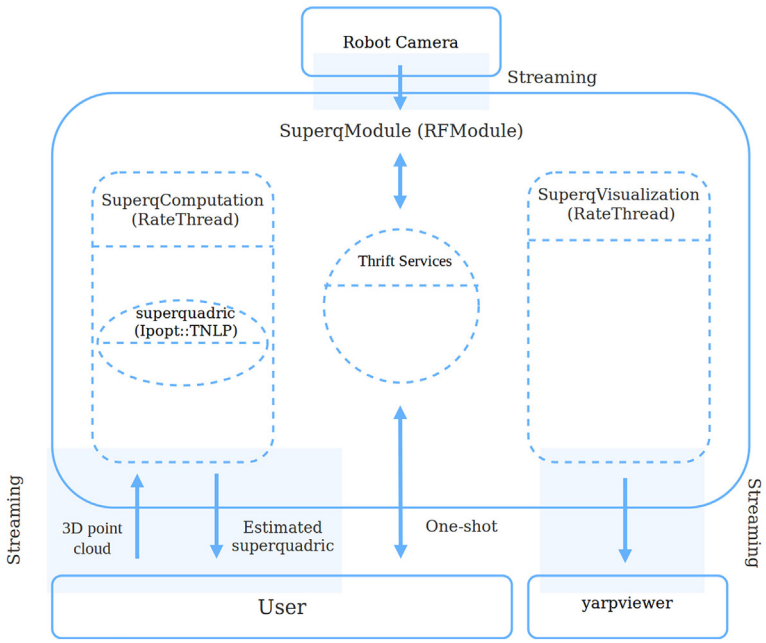

FIGURE 1 | Superquadric-model code structure. The class SuperqModule, derived from the YARP RFModule class, launches two threads, respectively for superquadric computation and visualization. The class provides some thrift services to the user for interacting with the module. More detail on the *user box* is provided in Section 3.3 and in **Figure 2**.

- If prior information is available on the object shape (e.g., given by a classifier or a vision recognition system), the module can use it to speed up the superquadric estimation. Particularly, if the object is labeled as *cylinder, box* or *sphere*, specific constraints can be used for improving the accuracy and reducing the execution time of the optimization problem.

The user can communicate with the *SuperqComputation* thread, through the *SuperqModule*, in the two different modes:

- In **streaming mode**—the 3D point cloud of the object should be sent to the module through a YARP *Buffered port* as a YARP *Property*. The user can access the current estimated superquadric through a dedicated YARP *Buffered port* as a YARP *Property*, where the main components of the superquadric are grouped as: *dimensions*, *exponents*, *center*, and *orientation*.
- In **one-shot mode**—the user can ask the module to compute the object superquadric by sending a single point cloud through a YARP *RpcClient Port* and getting a YARP *Property* including the estimated superquadric parameters as reply. In case the user asks for the superquadric filtered by the median filter, he should send a set of point clouds of the object in the same pose.

The superquadric computation, together with the superquadric filtering process, takes 0.1 s in average on Intel®Core™ i7-4710MQ Processor @2.50 GHz. This values is compatible with our real-time requirements.

3.1.2. SuperqVisualization

The visualization thread overlaps the estimated superquadric or the 3D points used by the optimizer on the camera image, for real-time visual inspection by the user (see **Figure 3** (4)). The average visualization time is equal to 0.01 s and can be enable or disabled by the user while the *SuperqModule* is running.

3.2. Superquadric-Grasping

The *superquadric-grasp* module implements the approach proposed in Vezzani et al. (2017) for the computation of grasping poses by using a superquadric modeling the object.

The *superquadric-grasp* module consists of the *GraspModule class*, derived from the YARP *RFModule class*. The *GraspModule* splits pose computation and visualization and grasp execution in three different classes:

- *GraspComputation class*, computing the pose for grasping the object;
- *GraspVisualization class*, showing the object model and the main information about the computed poses;
- *GraspExecution class*, which allows executing the grasping task once the pose is computed and one of the robot hand is selected.

As for the *superquadric-model* module, the *superquadric-grasp* implementation provides several *Thrfit IDL services* to the user to interact with the module and for getting information on the state of the module. The *superquadric-grasp* module structure is similar to the *superquadric-model* one, shown in **Figure 1**.

3.2.1. GraspComputation

This class handles the pose candidates' computation:

- Given the superquadric modeling the object, received as a YARP *Property* (see 3.1.1), the grasping poses for one or both

the hands (according to the user query) are computed together with a suitable trajectory by using the method proposed in Vezzani et al. (2017). The optimization problem is formulated and solved through the Ipopt C++ interface.
- The user can exploit some prior information for adapting the grasp computation to the desired scenario. In particular, the user can provide the module the height of the support on which the object is located (i.e., a table) to prevent the robot hand from hitting it. In addition, the constraints about the final hand pose can be modified according to the experimental scenario. For instance, the user can define the robot workspace by simply varying the variable upper and lower bounds of the optimization problem from the configuration files.

The pose computation process takes 2.0 s in average, which is consistent with the time requirements of a grasp task execution.

3.2.2. GraspExecution

The *GraspExecution class* controls the arm movements to accomplish the grasping task. In particular:

- The approaching step, i.e., the pose reaching through the trajectory waypoints, is executed through the YARP *Cartesian Interface* (Pattacini et al., 2010);
- Once the final pose is reached, the grasp is executed by using a precision grasp method described in Regoli et al. (2016) and available in the *Tactile Control library*.[5] The hand fingers close until the tactile sensors on the fingertips detect contact. Then, each finger is controlled to find a stable grasp for the object. Alternatively, the grasp can be performed by simply closing the fingers until a minimum pressure of the fingertips is measured. However, such an approach does not guarantee stability while lifting the object.

3.2.3. GraspVisualization

The visualization thread overlaps the computed poses and the received object superquadric on the camera image, for real-time visual inspection by the user (see **Figure 3** (5)). Some additional information, such as the volume graspable by the hand and the trajectory waypoints can be shown at the same time.

3.2.4. Communication with the Module

Unlike the *superquadric-model* framework, the user can communicate with the *GraspModule* only in **one-shot mode**. In particular, the user can query the module to:

- Compute the grasping poses and approaching trajectory, providing to the module the estimated superquadric of the object as a *Yarp Property* (as described in 3.1.1) and selecting one or both the hands. The solutions are given back to the user as a *Yarp Property*.
- Ask the robot to reach the final pose and grasp the object by selecting one robot hand. In the current code implementation, the robot performs a simple lifting test to check the stability of the grasp.

The additional *thrift services* allows setting on the fly parameters for grasp computation, visualization, and execution.

3.3. How to Use the Superquadric Framework

To use our grasping approach, the user is supposed to design a wrapper code to combine together the outcomes of the *superquadric-model* and *superquadric-grasp* modules. In addition, the implementation of a complete modeling and grasping pipeline requires the use of external modules for point cloud computation. We provide a tutorial code, which takes advantage of modules developed by the iCub community to achieve the modeling and grasping task. Hereafter, we report the main steps of the complete pipeline. The entire commented code is available on Github,[6] together with a detail description on how to run the code in the *README.md* file.

1. The object is labeled with a name through a recognition system.[7] The object label, together with information on its 2D bounding box, are stored by the *Object Property Collector*[8] (Moulin-Frier et al., 2017). The wrapper code is given the object name by the user (through a *RpcPort*) and uses it for asking the object property collector for the relative 2D bounding box.
2. The 2D blob of the object is computed by the *lbpExtract module*, once it is provided with the bounding box information. This uses Local Binary Pattern (LBP) (Ojala et al., 1996) to analyze the texture of what is in the robot view (a table in our experimental scenario). This texture is used for getting a general blob information both as an image, containing general white blobs of where the objects are, and as a *Yarp Bottle* containing lists of bounding box points. Then, the general blob information allow using grabCut algorithm (Rother et al., 2004) to properly segment all the objects on the table.
3. Given the 2D blob, the wrapper code reconstructs the 3D point cloud by querying the *Structure from Motion module* (Fanello et al., 2014). This module uses a complete Structure From Motion (SFM) pipeline for the computation of the extrinsics parameters between two different views. These parameters are then used to rectify the images and to compute a depth map.
4. Then, the wrapper code asks the *superquadric-model* to estimate the superquadric modeling the object by sending the acquired point cloud to the module.

```
Bottle cmd, superq_bottle;
//Fill the Bottle for querying
superquadric-model.
```

[5] https://github.com/robotology/tactile-control.

[6] https://github.com/robotology/superquadric-grasp-example.

[7] https://github.com/robotology/iol/tree/master/src/himrepClassifier.

[8] https://github.com/robotology/icub-main/tree/master/src/modules/objectsPropertiesCollector.

```
cmd.addString("get_superq");
Bottle &bottle_point = cmd.addList();
for (size_t i = 0; i < points.size(); i++)
{
 Bottle &in = bottle_point.addList();
 in.addDouble(points[i][0]);
 in.addDouble(points[i][1]);
 in.addDouble(points[i][2]);
}
superqRpc.write(cmd, superq_bottle);
//Then, extract the estimated superquadric from the Bottle superq_bottle.
```

5. Once the superquadric is estimated, the user code asks the *superquadric-grasp* module to compute pose candidates for grasping the object.

```
Bottle cmd, reply;
//Fill the Bottle for querying superquadric-grasp.
cmd.addString("get_grasping_pose");
//hand_for_computation can be "right", "left" or "both"
cmd.addString(hand_for_computation);
graspRpc.write(cmd, reply);
//Then, extract the grasping pose candidate from the Bottle reply.
```

6. Finally, the user can ask the *superquadric-grasp* to perform the grasping task.

```
Bottle cmd, reply;
//Fill the Bottle for moving the arm.
cmd.addString("move");
cmd.addString(hand_for_moving);
graspRpc.write(cmd, reply);
//The grasp is executed.
```

Figure 2 outlines the structure of the entire pipeline, following the steps described in this section. In **Figure 3**, we show some typical outcomes of all the steps described above. In addition, in the *README.md* files of the superquadric-model and superquadric-grasp repository, we provide two videos of the execution of the modeling and the grasping pipeline.[9]

[9]superquadric-model demo: https://www.youtube.com/watch?v=MViX4Ppo4WQ&feature=youtube. superquadric-grasp demo: https://www.youtube.com/watch?v=eGZO8peAVao.

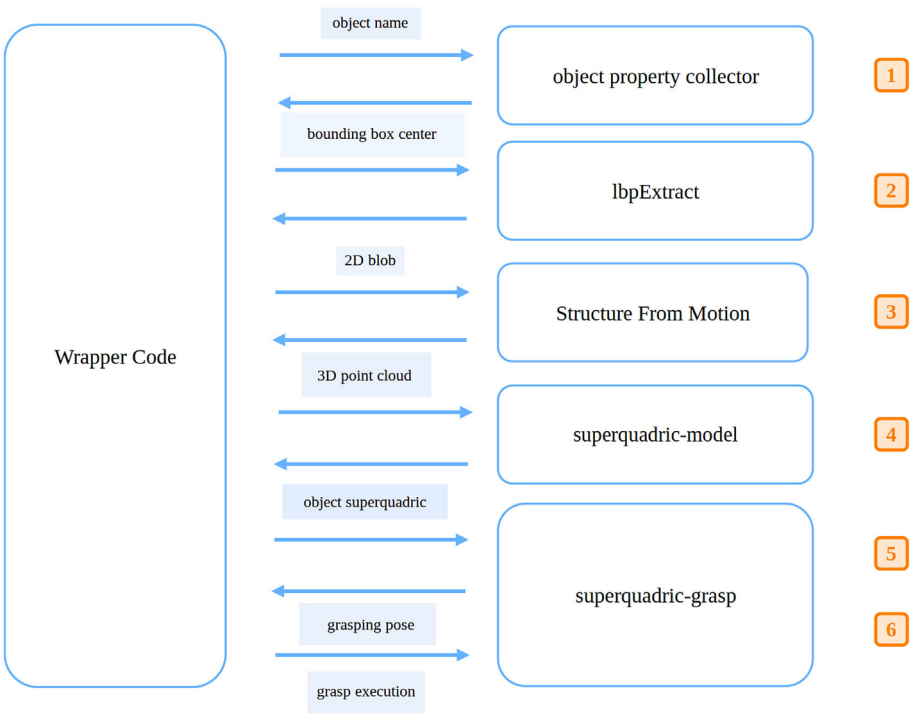

FIGURE 2 | Modules communication for the implementation of the modeling and grasping pipeline. The wrapper code manages the interaction between external modules and the *superquadric-model* and *superquadric-grasp* frameworks. Pipeline steps enumerated as in Section 3: (1) The wrapper code asks the object property collector for the bounding box information of the object. (2) Given that, lbpExtract module provides the 2D blob of the object. (3) The wrapper code sends the 2D blob of the object to the Structure From Motion module for getting the relative 3D point cloud. (4) The 3D point cloud is then sent to the *superquadric-model* for computing the superquadric modeling the object. (5) The wrapper code sends the estimated superquadric to the *superquadric-grasp* module, which computes suitable poses. (6) Finally, the *superquadric-grasp* is asked to perform the grasping task.

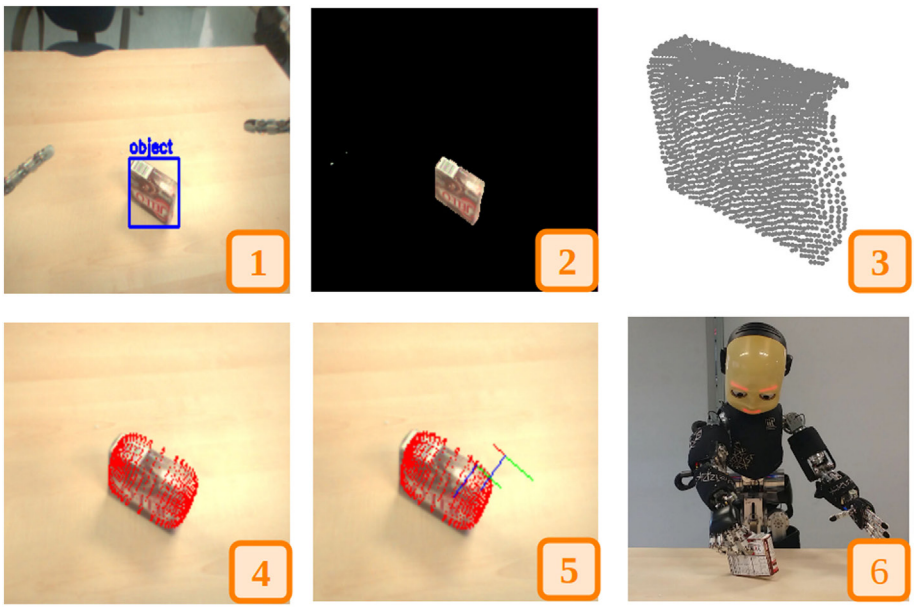

FIGURE 3 | Outcomes of the modeling and grasping pipeline. (1) The object is stored by the object property collector with the label *object*. (2) LbpExtract provides the 2D blob of the object. (3) The 3D point cloud is extracted from the disparity map, by querying the Structure From Motion module. (4) The superquadric modeling the object is reconstructed. (5) The grasping pose and approaching trajectory for the right hand are computed. (6) The robot grasps the object. (Steps (1), (2), (4), and (5) are represented by screenshots from the visualizers.).

4. KNOWN ISSUES

In this section, we report the limitations of our approach, together with possible solutions for facing them.

- Our approach is currently an open-loop approach. Once the object model and the grasping pose are computed, the robot reaches for the final pose without checking if the object pose changes. However, we could monitor the object pose, by estimating only the pose of the reconstructed superquadric - leaving its shape unchanged - with new point clouds while the robot is moving and until the object is in the robot field of view. This is a viable solution since our modeling approach is compatible with real-time requirements (as shown in Section 3.1).
- A further limitation caused by the open-loop nature of our approach is the missing compensation of errors between the robot stereo vision and system. To properly run the grasping pipeline, the user is required to properly calibrate the vision and the robot kinematics. In case errors between the two are still a problem for grasping the object, empirical offsets can be added for compensating for the errors. More information are provided in the *README.md* of the superquadric-grasp repository.
- A quite strong limitation of our approach is that it cannot automatically distinguish between good and wrong poses. For this reason, the user need to supervise the entire process and ask for a new model and pose in case the current outcome is not suitable for grasping the objects. In particular, this problem arises when the object cannot be represented with a single superquadric for its geometric shape. As future work, we aim at extend our approach for modeling more complex objects with multiple superquadrics.

5. CONCLUSION

In this work, we detail the implementation of the modeling and grasping approach pipeline described in Vezzani et al. (2017). We developed two modules, namely *superquadric-model* and *superquadric-grasp*, that respectively model objects through superquadric functions and computes suitable grasping poses for the iCub robot. Our leading idea was to develop a self-contained code that provides query services to the user. Our software handles only the information strictly necessary for the modeling and grasping approach and minimizes the dependencies from external modules. The user is supposed to design a wrapper code to combine together the outcomes of the two modules. We provide also an example of a external code in the *superquadric-grasp-example* repository for the implementation of a complete modeling and grasping pipeline.

In the next future, we would like to improve the approach we use for reaching the final grasping pose, which is a current limitation of our approach, as described in Section 4. The iCub proprioception is in fact affected by a number of impairments, mainly caused by elastic elements, which introduce errors in the computation of direct kinematics. Also, the iCub is provided with moving cameras for simulating the human oculomotor system. This makes the knowledge of extrinsic parameters and, thus, the object information estimation quite noisy. These sources of error might be crucial for grasping tasks, when a final pose is required to be reached with errors in order of 1 cm. We can solve this problem

by using the approach described in Fantacci et al. (2017), which provides a precise estimate of the robot end-effector pose over time and a visual servoing approach without the use of markers. Another extension of the modeling pipeline consists in using the recognition system[10] described in Pasquale et al. (2016) to classify the objects of interest according to their geometric property for using some prior information on their shape for improving and speeding up the superquadric estimation process, as mentioned in 3.1.1.

[10]https://github.com/robotology/onthefly-recognition.

AUTHOR CONTRIBUTIONS

GV developed the method and the code and described them in the manuscript. LN supervised the code and method development and the manuscript writing.

REFERENCES

Fanello, S. R., Pattacini, U., Gori, I., Tikhanoff, V., Randazzo, M., Roncone, A., et al. (2014). "3D stereo estimation and fully automated learning of eye-hand coordination in humanoid robots," in *2014 14th IEEE-RAS International Conference on Humanoid Robots (Humanoids)* (Madrid, Spain: IEEE), 1028–1035.

Fantacci, C., Pattacini, U., Tikhanoff, V., and Natale, L. (2017). "Visual end-effector tracking using a 3D model-aided particle filter for humanoid robot platforms," in *IEEE Conference on Intelligent Robots and Systems (IROS)* (Vancouver, Canada: IEEE).

Jaklic, A., Leonardis, A., and Solina, F. (2013). *Segmentation and Recovery of Superquadrics*, Vol. 20. Springer Science & Business Media.

Metta, G., Fitzpatrick, P., and Natale, L. (2006). YARP: yet another robot platform. *Int. J. Adv. Robot. Syst.* 3, 8. doi:10.5772/5761

Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., et al. (2010). The iCub humanoid robot: an open-systems platform for research in cognitive development. *Neural Netw.* 23, 1125–1134. doi:10.1016/j.neunet.2010.08.010

Moulin-Frier, C., Fischer, T., Petit, M., Pointeau, G., Puigbo, J., Pattacini, U., et al. (2017). Dac-h3: a proactive robot cognitive architecture to acquire and express knowledge about the world and the self. *IEEE Trans. Cogn. Dev. Syst.* doi:10.1109/TCDS.2017.2754143

Ojala, T., Pietikäinen, M., and Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions. *Pattern Recognit.* 29, 51–59. doi:10.1016/0031-3203(95)00067-4

Pasquale, G., Ciliberto, C., Rosasco, L., and Natale, L. (2016). "Object identification from few examples by improving the invariance of a deep convolutional neural network," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Deajeon, South Korea: IEEE), 4904–4911.

Pattacini, U., Nori, F., Natale, L., Metta, G., and Sandini, G. (2010). "An experimental evaluation of a novel minimum-jerk Cartesian controller for humanoid robots," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Taipei, Taiwan: IEEE), 1668–1674.

Regoli, M., Pattacini, U., Metta, G., and Natale, L. (2016). "Hierarchical grasp controller using tactile feedback," in *IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)* (Cancun, Mexico: IEEE), 387–394.

Rother, C., Kolmogorov, V., and Blake, A. (2004). Grabcut: interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.* 23, 309–314. doi:10.1145/1015706.1015720

Vezzani, G., Pattacini, U., and Natale, L. (2017). "A grasping approach based on superquadric models," in *IEEE International Conference on Robotics and Automation (ICRA)* (Singapore), 1579–1586.

Wächter, A., and Biegler, L. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.* 106, 25–57. doi:10.1007/s10107-004-0559-y

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

The reviewer, TF, and handling editor declared their shared affiliation.