

Technical Report

IRI-TR-16-01



Grasping Novel Objects

Nicola Covallero
Guillem Alenyà

May 2, 2016



Abstract

The work explained in this technical report is about evaluating some recent algorithms to grasp unforeseen objects for table clearance tasks. A tabletop object segmentation algorithm is proposed, and two recently published methods to generate grasps are discussed. The report presents practical considerations on how to use the segmentation algorithm and how to perform the tests with the evaluated algorithms for the generation of grasping poses. Finally, the results of both approaches and their comparison are discussed.

Institut de Robòtica i Informàtica Industrial (IRI)
Consejo Superior de Investigaciones Científicas (CSIC)
Universitat Politècnica de Catalunya (UPC)
Llorens i Artigas 4-6, 08028, Barcelona, Spain
Tel (fax): +34 93 401 5750 (5751)
<http://www.iri.upc.edu>

Corresponding author:
N Covallero
tel: +34 93 405 4490
ncovallero@iri.upc.edu
http:
[//www.iri.upc.edu/staff/ncovallero](http://www.iri.upc.edu/staff/ncovallero)

1 Introduction

1.1 Objectives

The main objective of this work is to find feasible grasping poses for novel objects for a table clearance application. The strategy to do that is first designing an algorithm to detect table top objects, next segment the tabletop objects and finally finding the grasping poses for those segmented objects. In detail, two algorithms have been evaluated for the grasping phase:

1. Antipodal Grasp Identification and Learning [4] (AGILE)
2. Height Accumulated Features [1] (HAF)

Both algorithms were published in 2015 and present interesting results. Moreover, the authors of both works made available the source code in ROS packages making them easy to test.

1.2 Report Structure

The report is structured as follows: first, the detection of tabletop objects and their segmentation are described in sections 2 and 3 respectively; second, the results with the **AGILE** and the **HAF** algorithm are discussed in sections 4.1 and 4.2 respectively; and finally, the conclusion in section 5. All the sections are provided of a brief theoretical introduction of the algorithms used, results on simulation and with real 3D images, and a detailed description of how to use the algorithms adapted to the IRI's framework.

1.3 Used technologies

The work has been done using the following technologies:

- Kinect
- ROS framework

All the work was validated both in simulation using Gazebo and then with real images using a Kinect available at the Manipulation and Perception Laboratory. In the laboratory, the Kinect is placed at the ceiling and its pose is facing down in a vertical direction pointing the table.

2 Tabletop Object Detection

An important part of the work is detecting the objects the robot has to interact with, and then detecting feasible grasping poses for each interested object. The strategy for the tabletop object detection phase is composed of 3 different steps:

1. **Table plane estimation** (by RANSAC): the points of the table are detected estimating first a plane in the point cloud, all the points which belong to such a plane are the points of the table.
2. **2D Convex Hull of the table**: having the points of the table a 2D convex hull is computed in order to get a 2D shape containing those points.
3. **Polygonal prism projection**: all the points are projected on the table plane previously estimated and all the points which projections belong to the 2D convex hull are considered to be points of tabletop objects.

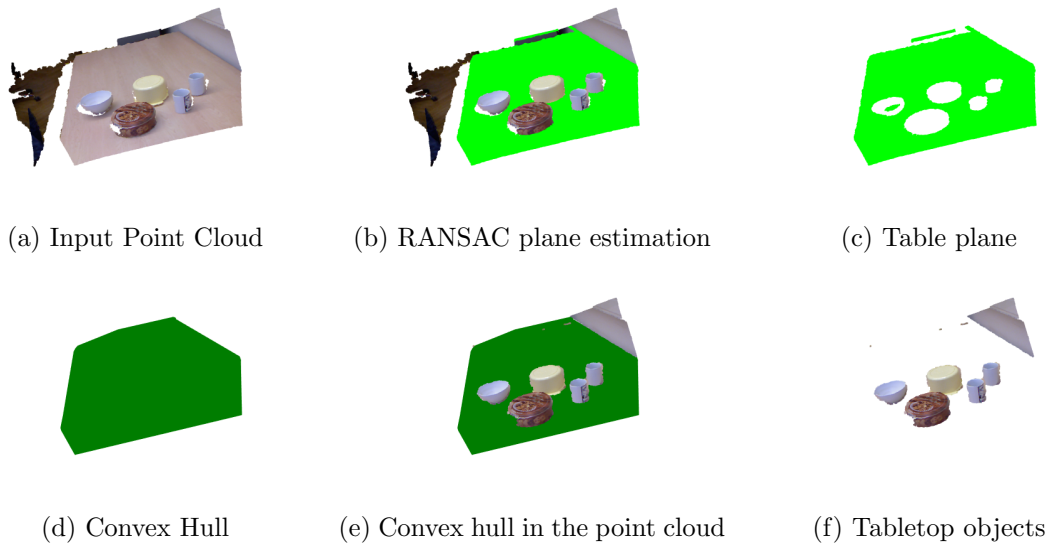


Figure 1: **Object Detection:** Given the point cloud (a), the estimated table's plane is obtained (b and c), its convex hull is extracted (d and e), and the tabletop objects are obtained by a polygonal prism projection (f).

The steps of this tabletop object detection algorithm are described in Figure 1 for the point cloud in Figure 1a.

The good point of this method is that it is able to detect the tabletop objects by first detecting the table's plane. Other planes, such as the one of the floor, could be present in the pointcloud and will be discarded since their projections on the table plane do not belong to the convex hull of the table plane.

Note that the point cloud in Figure 1a is composed of a table next to a wall, which is considered to be an object at the end of the object detection process. In the real set-up in the laboratory this will not happen since the table is located in the center of the laboratory.

3 Object Segmentation

Once the objects on the table are detected the following phase is to segment them in order to get a point cloud per object.

3.1 Supervoxel

For their segmentation the supervoxel concept is used. A supervoxel is a group of voxels that share similar characteristics.

In this work the supervoxels are computed with the Voxel Cloud Connectivity Segmentation (VCCS) algorithm [2], which was proposed in 2014 and gives good improvements with respect to previous state of the art methods, and still it is able to be used in online applications.

The algorithm works in 4 main steps:

- Voxelizing the point cloud
- Creating an adjacency graph for the voxel-cloud
- Creating seeds for the initial supervoxels centres

- Each seed is described by 39 features that describe spatial coordinates, colors and local surface model properties. Then a distance metric based on these features is defined.
- Clustering the voxels into supervoxels iteratively by means of the distance metric, the adjacency graph, and the search volume of the supervoxel.
- Once the search of all supervoxel adjacency graphs has been concluded, the centres of each supervoxel is updated by taking the mean of all its constituents.

This method has the good advantage that could be used on a point cloud formed by multiple observations, that is multiple depth sensors. This means that it can work on unorganized point clouds such the one obtained in section 2. The supervoxels applied to the point cloud of Figure 1a are shown in Figure 2.

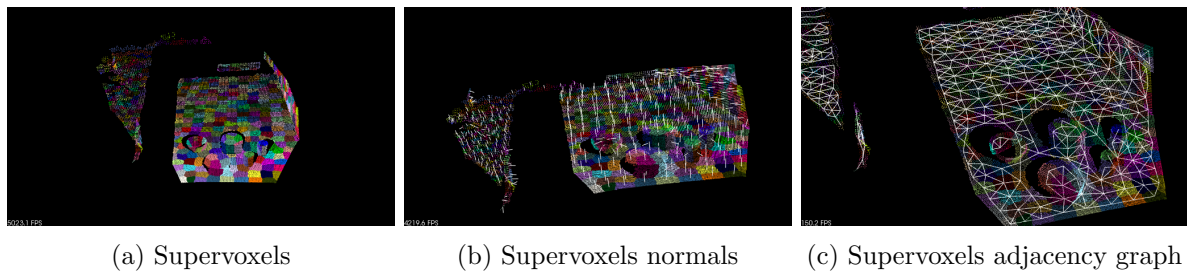


Figure 2: Supervoxels for the point cloud in Figure 1a computed by the VCCS algorithm

3.2 Segmentation

The following segmentation algorithm is proposed with the intention that should be also useful afterwards to detect how an object moved between two consecutive frames.

This segmentation is based on the assumption that the objects are convex. This is true for a large range of objects. Thus, knowing the supervoxels, their normals and the adjacency graph can be connected by looking for adjacent convex supervoxels. Considering such an assumption the object segmentation presented in this section is based on a 2 step strategy:

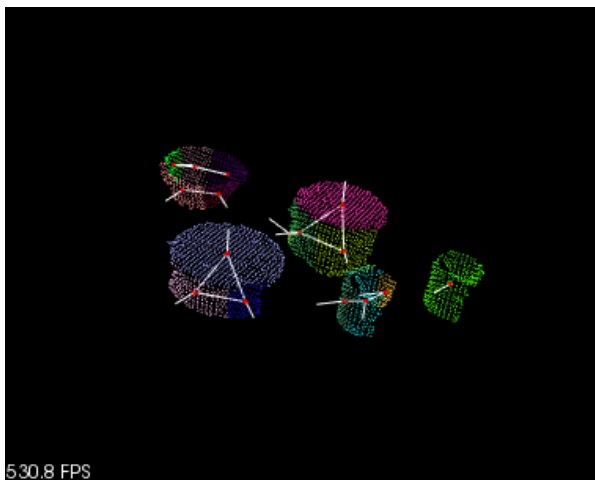
1. Clustering supervoxels in objects' facets and creating a facets adjacency map
2. Clustering adjacent convex facets into objects

Here facets are intended to be flat faces of an object.

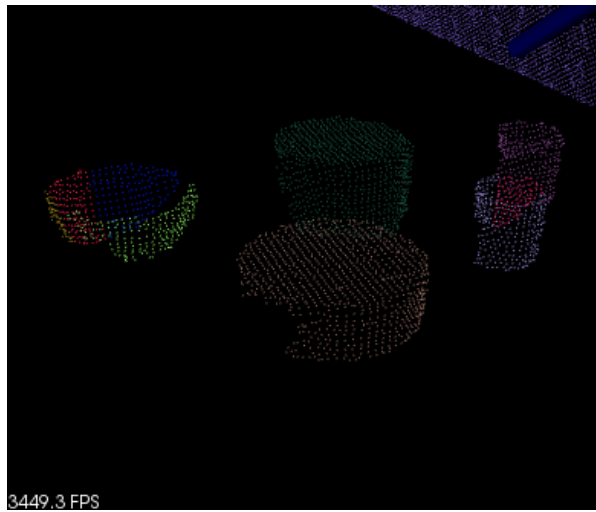
First, the algorithm clusters all the adjacent supervoxels which have a similar normal up to a threshold into facets. The facets concept plays an important role to detect how an object moved between two consecutive frames. With the proposed method the objects are formed by clusters of facets, so, considering an object in the first frame, to detect what is the correspondent object in the consecutive frame the facet of the object can be looked for in the second frame. If a facet in the first frame and another one in the second frame match, they belong to the same object, so the facets cluster they belong to is considered to be the same, that is the same object. The facets matching could be performed working in the RGB coordinates with the SIFT algorithm.

Then, to obtain the segmented objects, convex adjacent facets are clustered together. An example is shown in Figure 3.

There exist a lot of other segmentation strategies to do that, but in case of a cluttered scene where objects are not separated, the segmentations based on supervoxel are very useful, since the convexity criterion allows to segment colliding objects, as shown in Figure 4.



(a) Input Point Cloud

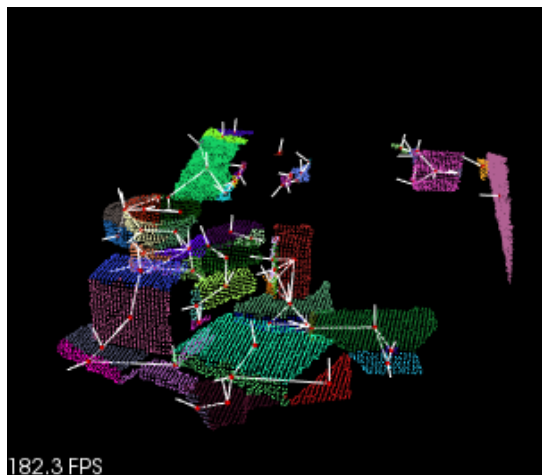


(b) RANSAC plane estimation

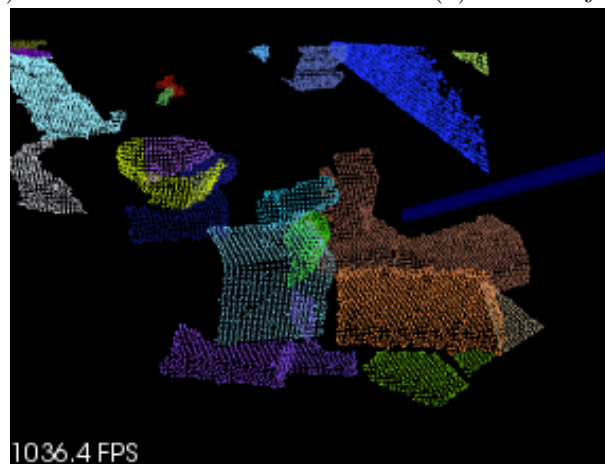
Figure 3: Segmentation for the scene of Figure 1a.



(a) Scene



(b) Facets adjacency graph



(c) Tabletop objects segmentation

Figure 4: Segmentation for the cluttered scene of Figure 4a

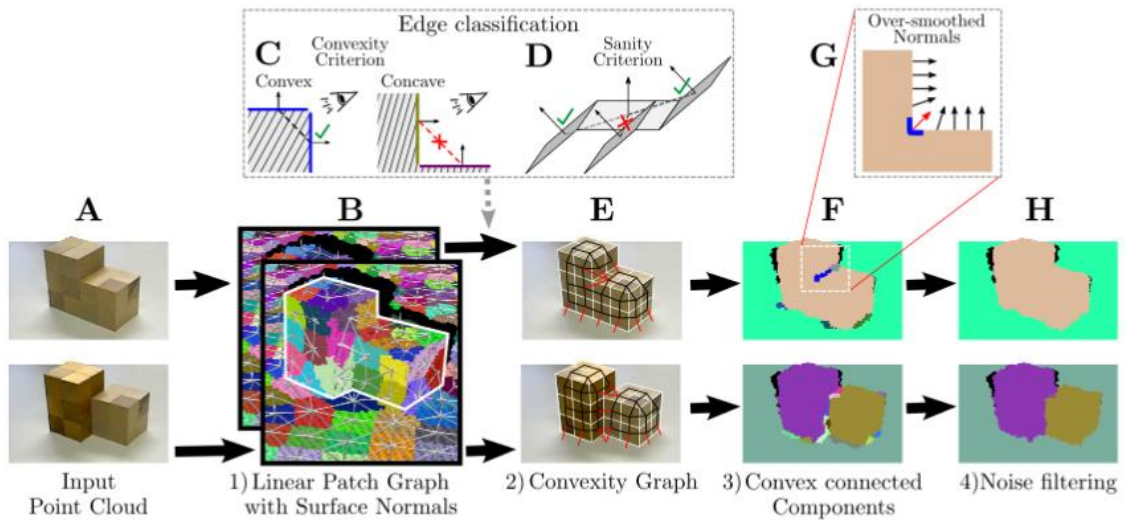


Figure 5: LCCP algorithm's structure. Reproduced from [3]

3.3 Local Convex Connected Patches Segmentation

During the development of the project, a very similar algorithm to the one above presented for the segmentation appeared, and its implementation was made available in the unstable version of the PCL library (PCL 1.8). The algorithm, called *Local Convex Connected Patches Segmentation* (LCCP) [3], has very easy structure which is described in the Figure 5.

It clusters all the adjacent convex supervoxels (patches), similarly to the previous proposed segmentation, using two criterion:

- Extended criterion: to consider two adjacent patches convex, both must have a connection to a patch which is convex with respect both patches
- Sanity Criterion: check if the adjacent patches which can be considered as convex present geometric discontinuities (see point D of Figure 5), in this case they are not considered as valid to form a cluster.

Then, due to the smoothed normals that could appear in some edges of the objects (point G Figure 5), the algorithm merges the clusters that are composed of few supervoxels to the biggest adjacent cluster.

This algorithm uses a more sophisticate technique than the segmentation algorithm proposed in section 3.2, and a fast and good implementation is also provided. The result of such algorithm in the scene in Figure 4a is shown in Figure 6.

It can be appreciated that the algorithm works very good in cluttered scene, being able to segment correctly most of the objects. Difficulties are encountered for objects that have curved surfaces, which are normally segmented in several objects. In this work we do not care about this since the tabletop object segmentation is indeed to provide an object to the grasping pose detector in order to grasp that object. If that object is actually a part of a bigger object, this means that the robot will grasp, and remove from the table, the big object which is our goal. In the next frame the new point cloud is segmented without taking care of the previous one.

¹https://github.com/PointCloudLibrary/pcl/blob/master/examples/segmentation/example_lccp_segmentation.cpp

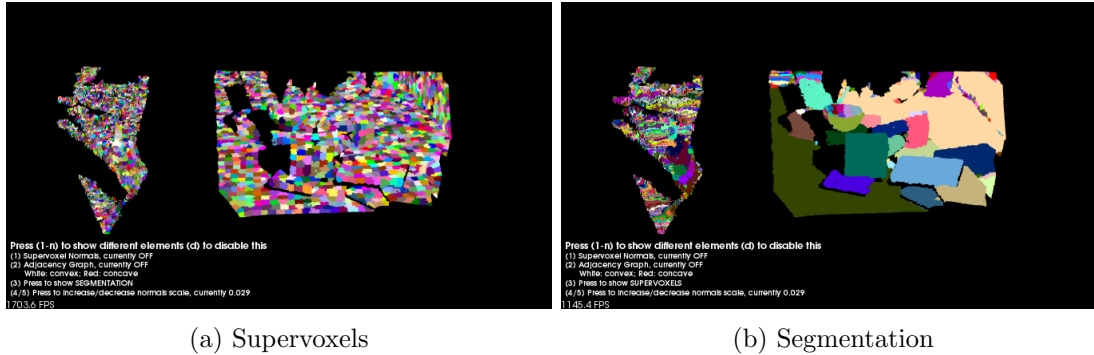


Figure 6: LCCP algorithm applied to the scene in Figure 4a with default parameters. These images were reproduced from the PCL’s example code of the LCCP segmentation¹.

3.4 Algorithm Implementation

The Tabletop Object Segmentation algorithm (called `tos_supervoxels`) implemented performs the following sequence of operations:

1. table top object detection
2. object segmentation (by `VCCS`² and `LCCP`³ algorithms, both available in the PCL 1.7 and 1.8 libraries respectively).

It has before been implemented in C++ and then in ROS as a service.

The algorithm’s implementation is quite fast because the LCCP segmentation is directly applied to the tabletop objects, not to the input pointcloud, thus the pointcloud it has to work with is quite reduced and the processing of those few points is fast, usually around 0.5 seconds.

As input the algorithm needs only the pointcloud that the user wants to segment, and eventually some parameters different from the default ones. These parameters are:

- Parameters for the supervoxels[2]:
 - `disable_transform`: boolean variable to disable the single-view transform (this is the default for unorganized clouds, only affects organized clouds).
 - `voxel_resolution`: double variable to set the voxel size, which determines the leaf size of the underlying octree structure (in meters).
 - `seed_resolution`: double variable to set the seeding size, which determines how big the supervoxels will be (in meters). Bigger it is more likely different objects are segmented as one.
 - `color_importance`: double variable to set the weight for color - how much color will influence the shape of the supervoxels.
 - `spatial_importance`: double variable to set the weight for spatial term - higher values will result in supervoxels with very regular shapes (lower will result in supervoxels which follow normals and/or colors, but are not very regular).
 - `normal_importance`: double variable to set the weight for normals - how much surface normals will influence the shape of the supervoxels.

- Parameters for the LCCP segmentation[3]:

²http://docs.pointclouds.org/1.7.1/classpcl_1_1_supervoxel.html

³http://docs.pointclouds.org/trunk/classpcl_1_1_l_c_c_p_segmentation.html

- `concavity_tolerance_threshold`: double variable to set the threshold used to consider two adjacent supervoxels as convex.
 - `smoothness_threshold`: double variable to set the threshold to consider two supervoxels as unsmooth.
 - `min_segment_size`: integer variable to set the minimum number of supervoxels needed to form a valid cluster. A cluster formed by less supervoxels than `min_segment_size` is merged to the bigger adjacent one. This parameter has to be chosen accordingly to the `seed_resolution`.
 - `use_extended_convexity`: boolean variable to use or not the extended convexity criterion.
 - `use_sanity_criterion`: boolean variable to use or not the sanity criterion.
- Parameters for the tabletop object detection explained in section 2:
 - `zmin`: double variable to set the minimum distance orthogonal to the table plane to consider a point as a point of a tabletop object (This works as the threshold for the RANSAC algorithm). For tables with a not very flat surface this parameter has to be higher.
 - `zmax`: double variable to set the maximum distance orthogonal to the table plane to consider a point as a point of a tabletop object. Just set it to be high enough (1 meter or 2) to consider all the possible objects points.
 - Extra parameter to filter small objects out:
 - `th_points`: integer variable to set the minimum number of points needed to consider a cluster as a valid object. This parameter is useful to filter small isolated clusters that could appear. To work with small objects this parameter has to be low.

The algorithm needs as input a point cloud of type `pcl::PointXYZRGBA` and it returns the segmented objects either as a vector of `Object` struct or a `pcl::PointCloud<pcl::PointXYZRGBA>` vector. The `Object` struct is defined as:

```
struct Object
{
    pcl::PointCloud<pcl::PointXYZRGBA> object_cloud;
    int label;
};
```

where `label` is the label assigned by the LCCP algorithm.

An example of how to use the code, once included the library in the project, is:

```
tos_supervoxels segmentation;
segmentation.init(*cloud);
segmentation.segment();
std::vector<pcl::PointCloud<pcl::PointXYZRGBA> > segmented_objs;
segmented_objs = segmentation.get_segmented_objects_simple();
segmentation.reset();//to free the memory
```

In order to manage in an easy way all the parameters of the algorithm, the library also comes with the class `tos_supervoxels_parameters`. This class just has the constructor which initializes all the parameters to their default value, and these parameters are defined as public members.

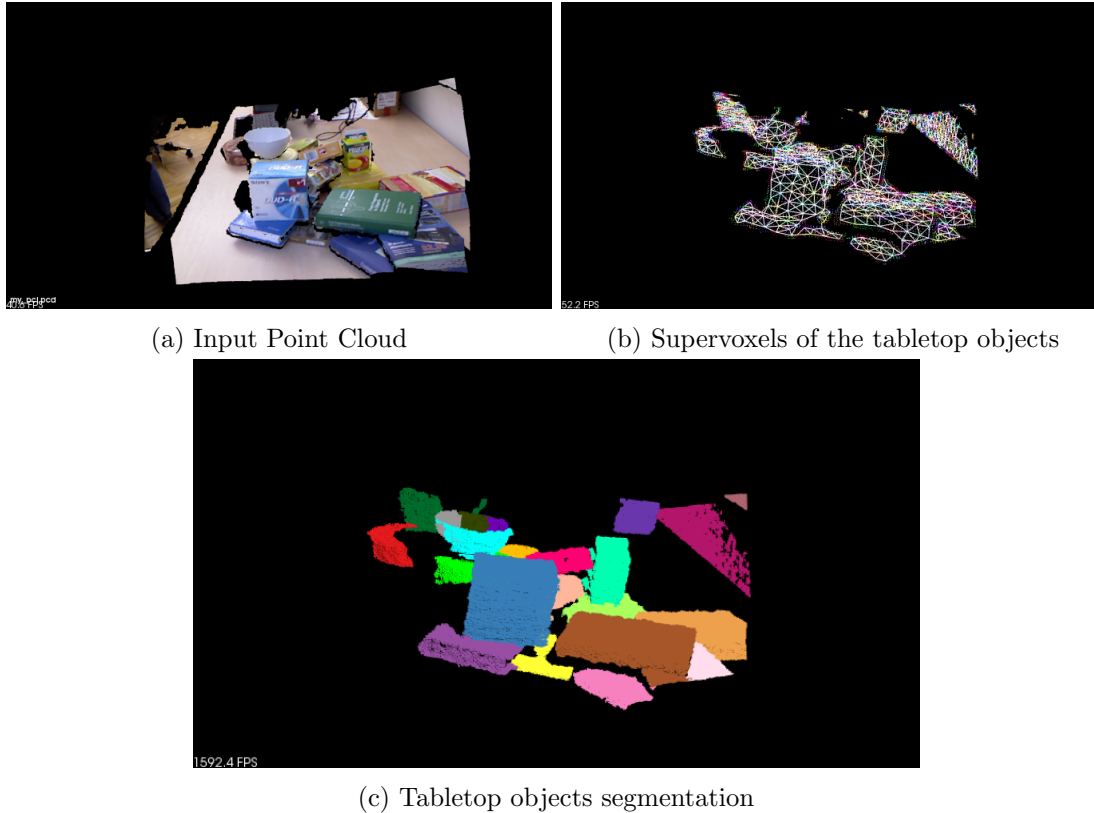


Figure 7

The algorithm also has some functions to plot in a PCL visualizer the supervoxels, the adjacency map and the results of the segmentation for debugging. The library provides a useful example which use the most important functions of the algorithm. Such an example also shows how to retrieve the points in the Kinect's RGB image in case the user need to know which are the points of a certain object in the image. To do this a KdTree is necessary which searches in the original point cloud which is the nearest point to the object's point considered. This operation is still quite fast and an example is shown in Figure 8.

Another possible implementation of the algorithm is the following:

1. object segmentation (by LCCP algorithm)
2. table top object detection.

In this case the algorithm has the capability to perform the objects segmentation knowing the indices of their points in the original point cloud. The computational cost is higher because the LCCP is applied to the whole cloud and its computational cost is similar to the proposed implementation plus the KdTree. In the other hand it has a worst segmentation result due to the fact that the LCCP algorithm could merge some supervoxels that actually belong to an objects to the cluster associated to the table's plane. At the moment to consider only the points that stand above the table, the result is an object composed by supervoxels that easily are not directly connected, they were connected through the table's supervoxels which has been removed. Such a problem is shown in Figure 9.

Download and compile Download the repository

```
$ git clone https://gitlab.iri.upc.edu/perception/tos_supervoxels.git
```



(a) RGB image of the scene

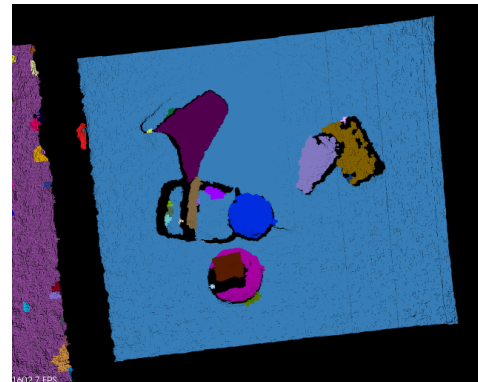


(b) Example of a detected object in the RGB image.

Figure 8: Objects segmentation in the RGB image, after retrieving the segmented objects points in the original point cloud with KdTree.



(a) RGB image of the scene



(b) Segmentation of the scene

Figure 9: In this case the hands of the jug and the pan are merged to the table's plane, so at the moment to take off the plane and considering only the points above such a plane, the two hands will results as an unique object although they are far.

in the `laborobotica/algorithms` folder (if you are using the IRI folder structure). Compile using:

```
cd tos_supervoxels/trunk
mkdir build && cd build
cmake ..
make
sudo make -j2 install
make doc
```

Then installation is needed for the ROS package of the algorithm.

Dependencies The algorithm has the unique dependence of PCL 1.8, which at this moment has to be compile from sources ⁴. The example also depends on OpenCV, which should be already installed in a system with a standard installation of ROS.

⁴http://pointclouds.org/documentation/tutorials/compiling_pcl_posix.php

Use the library In the folder `/src/examples` there is an example and its `CMakeList.txt` that shows how to link the library and how use it in a `.cpp` code. To run the example go to the `bin` folder and run:

```
$ ./tos_supervoxels_test my_pcl.pcd
```

For the help just launch:

```
$ ./tos_supervoxels_test
```

a menu with all the options to change the algorithm's parameters will appear.

If you want to see the original point cloud, launch in another terminal:

```
$ pcl_viewer my_pcl.pcd
```

NOTES Due to the fact that the version of the Point Cloud Library the algorithm is based on is still in development, the sources code the user uses to compile the library could have to be modified, this would affect only the LCCP algorithm, and its correction should involve just few lines of code.

3.5 ROS implementation

The algorithm has been implemented in ROS, in the package `iri_tos_supervoxels`, as a service called

```
/iri_tos_supervoxels/segment_objects
```

and defined as:

```
sensor_msgs/PointCloud2 point_cloud
---
iri_tos_supervoxels/segmented_objects objects
sensor_msgs/PointCloud2 segmented_cloud
```

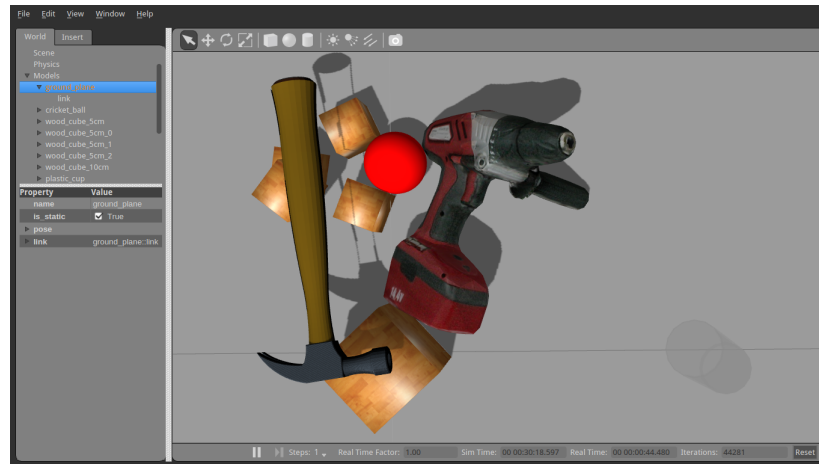
where `point_cloud` is the point cloud of the scene. The response of the server is `objects`, which is of type `iri_tos_supervoxels/segmented_objects`, that is a vector of the `sensor_msgs/PointCloud2` message, and `segmented_cloud` which is a point cloud composed only by the detected objects and to each object is assigned a random color (useful for debugging or have a visual result of the segmentation).

Download and compile The package `iri_tos_supervoxel` needs the `tos_supervoxel` library to be installed in the system, it has the same dependencies of the `tos_supervoxel` library and the also the package `iri_core`⁵ has to be installed. To install the `iri_tos_supervoxel` launch the following command in `iri` workspace:

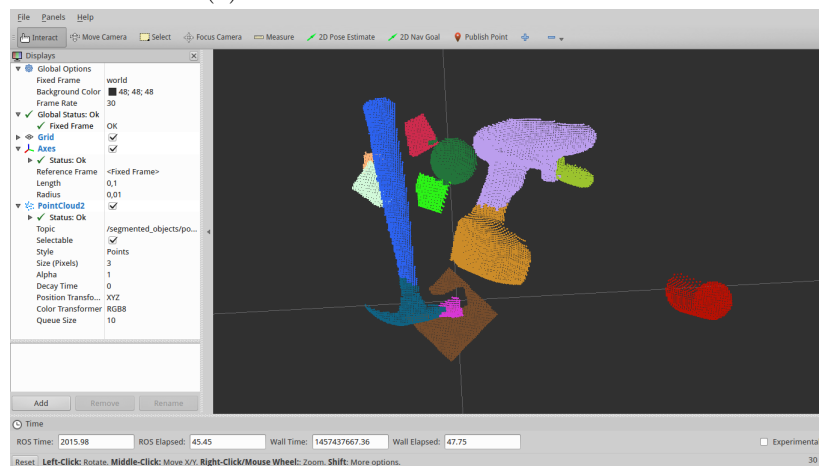
```
$ cd src/
$ git clone https://gitlab.iri.upc.edu/perception/iri_tos_supervoxels.git
$ cd ..
$ catkin_make --only-pkg-with-deps iri_tos_supervoxels
```

If the package `iri_core` is not installed, launch the following commands in the `iri` workspace:

⁵https://devel.iri.upc.edu/pub/labrobotica/ros/iri-ros-pkg_hydro/metapackages/iri_core



(a) Gazebo simulated cluttered scene



(b) Segmentation of the scene

Figure 10: Validation of the algorithm in simulation.

```

$ cd src/
$ svn checkout
https://devel.iri.upc.edu/pub/labrobotica/ros/iri-ros-pkg_hydro/metapackages/iri_core
$ catkin_make --only-pkg-with-deps iri_core
  
```

In the package `iri_tos_supervoxels` there is also an example of a client node which calls the service and publishes the segmentation results and the segmented cloud for a visual result. The algorithm has been validated in simulation in Gazebo simulating a Kinect and a cluttered scene as the one in Figure 10a, the equivalent segmentation is shown in Figure 10b. To simulate the Kinect you can clone the following package with git:

```
$ git clone https://github.com/NicolaCovallero/kinect_sim.git
```

Such a package does not need to be compiled.

To launch the simulation launch in different terminals the following commands:

```

$ roscore
$ roslaunch kinect_sim kinect.launch
$ roslaunch iri_tos_supervoxels tos_supervoxels.launch
$ roslaunch kinect_sim rviz.launch
  
```

To use it with the real Kinect launch in different terminals the following commands:

```
$ roscore
$ roslaunch kinect_sim real_kinect.launch
$ roslaunch iri_tos_supervoxels tos_supervoxels.launch
$ roslaunch kinect_sim rviz.launch
```

4 Grasps Detection

In this section some methods of the current state of the art have been tested in order to find a suitable algorithm for the goal. The majority of the methods of the states of the art which works fine are usually based on a database of objects, so the algorithm usually tries to detect which is the category of the objects and then it already knows what are the feasible grasping poses for each object in its database. The past year were released two interesting ROS packages that deal with unforeseen objects for grasping poses detection. Both are based on a support vector machine, the point cloud is described as features and given as input to such a support vector machine, then some feasible grasps are detected if possible.

In this section the two algorithms (**AGILE** and **HAF**) are introduced and the results are presented and commented. Both algorithm were designed to work with a 2-finger parallel gripper.

4.1 AGILE

The Antipodal Grasping Identification and LEarning algorithm (**AGILE**) [4] is based on detecting grasping through geometric features of a point cloud. In particular it is based on detecting antipodal points, and with them detecting feasible grasping poses related to such antipodal points.

The algorithm works with the following sequence of steps:

- Voxelizing point cloud
- Applying workspace limits: to consider only an interested region of the scene
- Sample some random points to the voxelized point cloud
- For those points the algorithm estimates the Darboux frame⁶
- Sample some orientations and positions for the gripper coherent with the estimated Darboux frame
- If the closing region⁷ of sampled grasping poses contains points in the point cloud it is added to the hypothesis.
- Validation of the hypothesis by a support vector machine.

As features to describe the grasping poses the histogram of gradients of the points in the closing region of the gripper is used. The support vector machine has been trained with two Kinects. In particular the authors trained it with the point cloud of one Kinect, with the point cloud of the second one and with both, so the support vector machine should be able to predict the occluded shape of the objects when working with a single Kinect.

The authors provided a ROS package for the algorithm⁸. It simply needs as input a point cloud and the output is a vector of the **Grasp** message defined as:

⁶Any frame aligned with the surface normal is a Darboux Frame

⁷Planar region between the two fingers of the gripper

⁸http://wiki.ros.org/agile_grasp

```
geometry_msgs/Vector3 center
geometry_msgs/Vector3 axis
geometry_msgs/Vector3 approach
geometry_msgs/Vector3 surface_center
std_msgs/Float32 width
```

where **center** is the center point of the object where the gripper will grasp the object, **axis** is the vector orthogonal to the closing plane of the gripper, **approach** is the approaching direction of the grasping pose, **surface_center** is the center point projected on the surface of the object along the approaching direction and finally **width** indicates the estimated width of the gripper when it is closed (i.e. it grasps the object).

In this work also a small ROS package has been developed in order to filter some grasps and to improve the visualization in Rviz.

Download and compile This package, called `agile_grasp_filter`, can be downloaded by typing in a terminal, inside the `/src` folder of a ROS workspace:

```
$ git clone https://github.com/NicolaCovallero/agile_grasp_filter.git
```

Such a package has the dependence to the `agile_grasp` package, to download it run in the `/src` folder of your working space:

```
$ git clone https://github.com/atenpas/agile_grasp
```

Then compile the two package:

```
$ catkin_make --only-pkg-with-deps agile_grasp
$ catkin_make --only-pkg-with-deps agile_grasp_filter
```

The `agile_grasp_filter` package filters out the grasp detected by the AGILE algorithm that are not perpendicular to the table up to a threshold. In particular the depth axis of the Kinect is supposed to be orthogonal to the table plane, and the threshold is the cosine of the angle between the approaching vector of a certain grasp and the depth axis of the Kinect. The filtering was thought in order to consider only grasps that can be performed easily by the robot, such as the vertical ones.

The package will publish in a new topic the filtered grasping poses and some markers to improve the Rviz visualization, such as the marker related to the grasps considered as valid and the closing gripper direction.

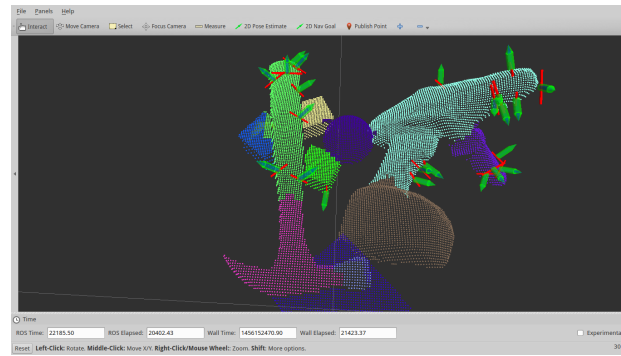
The most important parameters of AGILE algorithm are the ones to model the gripper, and among them the most important is the maximum opening width of the gripper (`hand_outer_diameter`). Also the number of samples could be chosen and this affects the number of points that are sampled to find grasping hypothesis (`num_samples`).

To launch the simulation launch the following commands in different terminals:

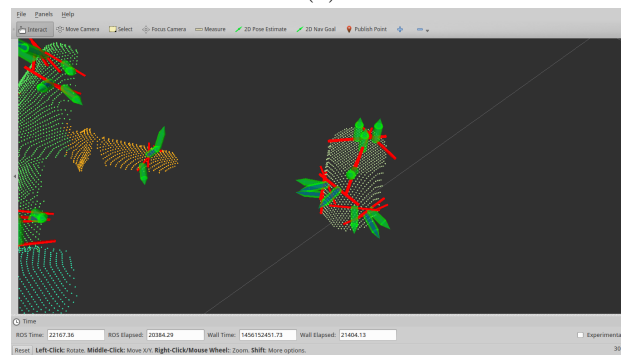
```
$ roslaunch kinect_sim kinect.launch
$ roslaunch iri_tos_supervoxels tos_supervoxels.launch
$ roslaunch agile_grasp_filter agile_grasp.launch
$ roslaunch agile_grasp_filter rviz.launch
```

To launch the nodes with the real sensor launch the following commands in different terminals:

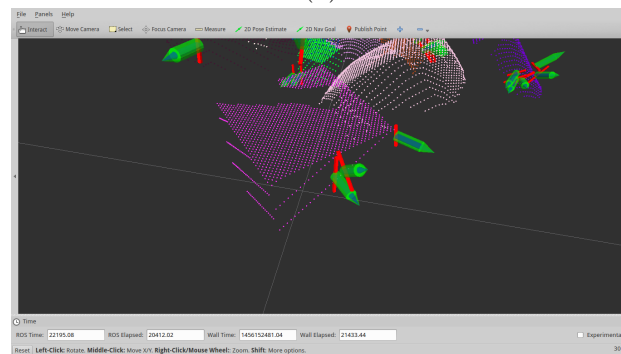
```
$ roslaunch kinect_sim real_kinect.launch
$ roslaunch iri_tos_supervoxels tos_supervoxels.launch
$ roslaunch agile_grasp_filter agile_grasp.launch
$ roslaunch agile_grasp_filter rviz.launch
```



(a)



(b)



(c)

Figure 11: AGILE algorithm in simulation. The green markers show the approaching direction and the red markers show the gripper closing direction.

Notes In this case the object segmentation is not useful since we worked directly on the whole point cloud. This has the advantage that the agile algorithm will take into account how the cluttered scene is in order to find feasible grasps. In the other hand to grasp a certain object we should first detects all the feasible grasps and then for each grasp detect which is the object it is related with. Applying the AGILE algorithm for each object separately it will detect some poses that are not feasible because of collision with the other objects that would be not taken into account.

We can observe also that the width (the length of the red markers is equal to the width) usually is very small, therefore its better to not take into account this parameter and consider that the gripper has to be open with its maximum width for the grasping. In this case it should be checked that there are no collisions with the environment.

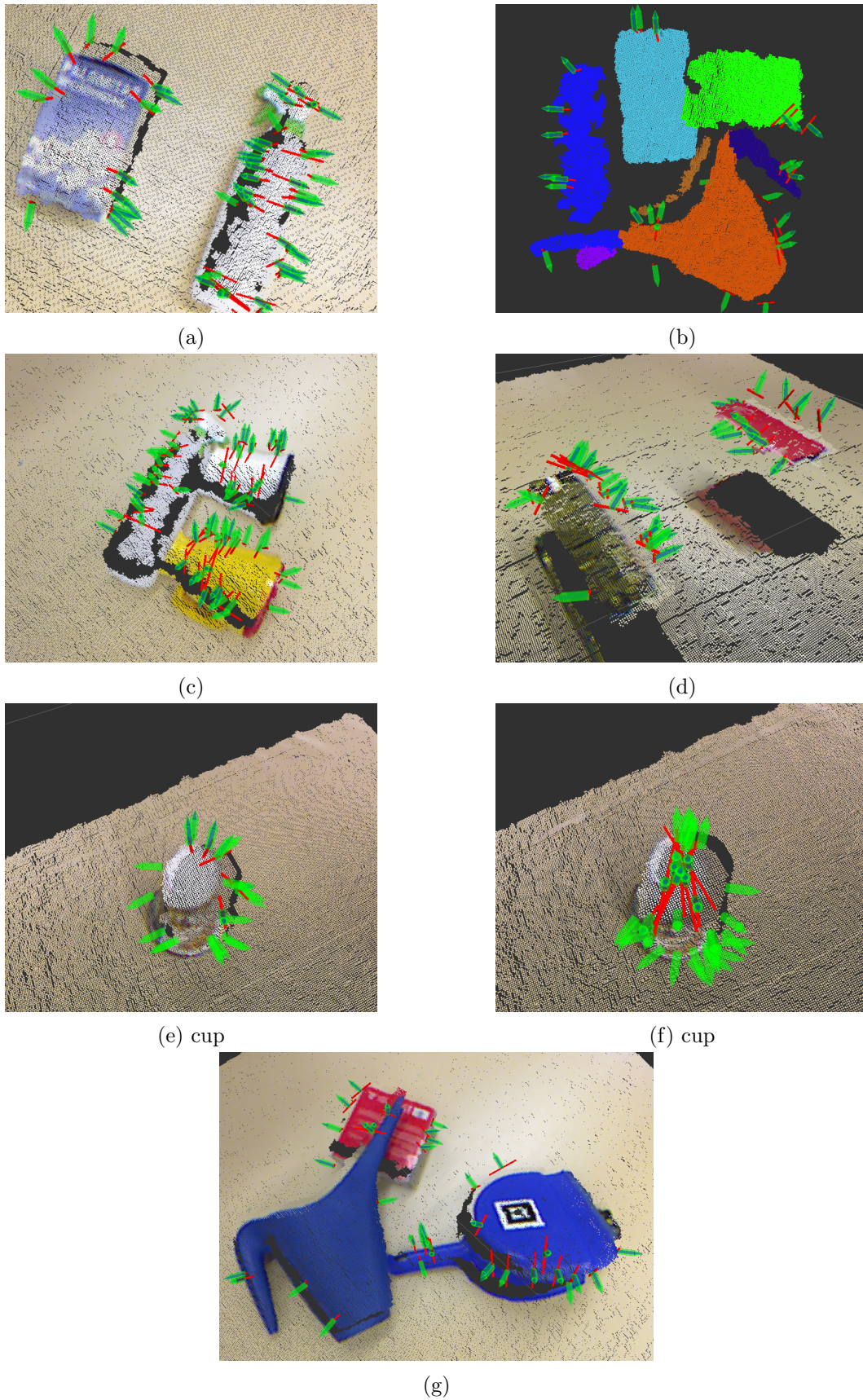


Figure 12: AGILE algorithm in a real scene. The green markers show the approaching direction and the red markers show the gripper closing direction for the filtered ones. Only for Figures 12e and 12f the filtering is applied, for the others images all the grasping poses are considered as valid ones.

Analysis The algorithm has been validated in simulation and some results are shown in Figure 11. In Figure 11a we can observe that the majority of the grasping poses are feasible ones, as for Figure 11b. A bad case is for the one in Figure 11c, the Kinect has a partial view of the box's side and the algorithm considers those few points as a complete side, it sees that object like a sheet. The support vector machine fails in estimating the occluded sides, this could be due to the way the authors trained their support vector machine, that is the Kinect's pose with respect to the table. This is a clear example of why the filtering has been implemented. In particular it has been seen that such strange behaviour appears for occluded sides, therefore a simple technique is the one to avoid approaching directions that are not orthogonal to the table plane and which could be associated to this case. This also helps the robot to perform the table clearance task since it is easier, more likely successful, performing vertical grasps.

In Figure 12 some results with some real scenes are shown. For cylindrical objects the grasps are almost all feasible, but for the boxes, since only the top side is seen by the Kinect, the support vector machine treats that objects as one with small depth, and therefore it likely returns grasps at its sides, which are not feasible grasps.

An interesting example of how the parameter `hand_outer_diameter` affects the results is shown in Figures 12e and 12f, where it was respectively set to *0.09meters* and *0.15meters*. Figure 12e represents the same problem for the boxes and we can observe that increasing the `hand_outer_diameter` value and coupling this strategy with filtering in order to consider only vertical approaching directions resolves partially the problem.

Another thing to point out is the difference of the quality of the results between the simulation and the real tests. This is due to the noise of the real sensor which is not able to measure properly the depth at the edges.

Further Work Ten Pas and Platt, in his paper about AGILE [4] do not clarify which was the pose of the Kinect with respect to the working area. Due to its strange behaviour in the occluded sides, their Kinect's pose is likely different to the one in our laboratory and probably it was able to get at least two sides (top and one lateral side) of the objects. The ROS package also comes with a node to train your own support vector machine, but the documentation of the package does not allow to understand properly how it works. The training node needs several point clouds to train the classifier, but it is not specified how they have to be, that is if they have to include also the table or not, and how the labelling of positive and negative samples is done.

As further work, a classifier with the Kinect available in the laboratory could be trained in order to avoid the problem of the occluded sides. Another naive approach could be filling the occluded sides of the table top objects in the orthogonal direction of the table plane. Finally, a score function based on the approaching orientation and the width could be designed in order to give a score to each possible grasping pose.

4.2 HAF

The Height Accumulated Features (HAF) [1] is a grasping pose detection algorithm based on the topographic features of the objects, in particular on the height of the points, and it uses a support vector machine to classify possible grasping poses.

It basically works dividing the working zone in a grid cell and computing the features as a weighted sum of the medium height of the points of each cell. These features are computed for each possible closing direction of the gripper, in a discrete space, for a certain approaching direction. The features have been created randomly (up to 71000) and the best 300 have been chosen through the F-score metric⁹. Then to choose the grasp it uses a weighting system which

⁹Technique which measures the discrimination power of features

assigns a score to each possible grasping pose. The score is a function of how many possible grasps are possible near the considered point.

This operation is for only a given approaching direction of the gripper. To test more possible approaching directions the point cloud has to be transformed accordingly to the orientation and the algorithm is repeated for the transformed point cloud.

The authors implemented a ROS package¹⁰ in which the algorithm is implemented as a service, and there is an example of a client node that calls it giving the point cloud where the user wants to look for grasping poses. The results relative to the best grasping pose is published as a string in a topic.

In this work another client node (called `calc_grasppoints_client_sim`) has been implemented, in a forked version of the original package to make easier the implementation, in order to detect grasping poses for each segmented object. Such a client performs the following steps:

1. Receiving a point cloud from the Kinect.
2. Calling the object segmentation service.
3. Transforming the point cloud of each object with respect the base reference frame.
4. Computing the search area for each object (this area will define the grid cell): this is computing by looking its maximum and minimum x and y dimensions, and centring the area to the object.
5. Detecting the best grasp for each object for only the approaching direction orthogonal to the table. (This is done by calling the service provided by the authors in the package¹¹)
6. Publishing markers to visualize the grasping poses
7. Publishing the grasps as a vector of the `Grasp` message defined as

```
float64 val
geometry_msgs/Point gcp

geometry_msgs/Point gp1
geometry_msgs/Point gp2

geometry_msgs/Vector3 approach_vector
float64 roll
```

where `val` is the value for the grasping pose, `gcp` is the central grasping point, `gp1` and `gp2` are the grasping points at the sides of the object, `roll` is the rotation around the vertical axis (this defines the closing direction for a certain approaching direction).

Download and compile

The adapted package with the cited client node (`calc_grasppoints_client_sim`) can be downloaded by the following command in the source folder of the working space:

```
$ git clone https://github.com/NicolaCovallero/haf_grasping
```

Then, to compile it, launch in the root of the workspace the following command:

```
$ catkin_make --only-pkg-with-deps haf_grasping
```

¹⁰http://wiki.ros.org/haf_grasping

¹¹http://wiki.ros.org/haf_grasping

Once compiled, to use it in a simulated environment type in different terminals the following commands:

```
$ roslaunch kinect_sim kinect.launch
$ roslaunch iri_tos_supervoxels tos_supervoxels_server.launch
$ roslaunch haf_grasping rviz.launch
$ roslaunch haf_grasping haf_simulation.launch
```

To work with the real kinect launch the following:

```
$ roslaunch kinect_sim real_kinect.launch
$ roslaunch iri_tos_supervoxels tos_supervoxels_server.launch
$ roslaunch haf_grasping rviz.launch
$ roslaunch haf_grasping haf_simulation.launch
```

where the `real_kinect.launch` just contains an include to launch the `freenect_launcher` and static transformation to add the reference frame of the footprint of the WAM robot. In this case since it has been observed that the points of the table measured by the Kinect have a height less than 0, to the reference frame has been added an offset in z-axis just to make all points have a height greater than 0, in order to avoid problems. This because the service node by default uses another reference frame `tf_help` to do its computations, and it needs to have all the points with a positive height with respect that frame. This frame is centred to the search area (defined with respect the reference base frame) with a small translation in the negative direction of the z-axis of the base frame.

The HAF node server node (the original one) also publishes some markers:

- Grey area: define the area which is searched for potential grasps
- Long red line: it indicates the closing direction which is currently tested
- Red spots: they indicate the positions where the grasps are tested
- Green bars: indicate identified potential grasps available. The height of the bars indicates the grasp evaluation score (the higher the better)
- Black arrow: indicates the best grasp position found

The `calc_grasppoints_client_sim` node moreover publishes:

- Green markers: approaching direction for the best grasp for each object
- Small red line: closing direction for the best grasp for each object

The `calc_grasppoints_client_sim` asks the service to detect the best grasp for each object, it gives as input the object's point cloud retrieved by the segmentation step, without taking into account other possible adjacent objects. This has the drawback that once the best grasp have been detected, they have to be checked for collision. In the other hand, if to the service the points of adjacent objects are also given as input, the algorithm could detect a best grasp which belongs to an adjacent object to the interested one.

In the case the goal is performing a table clearance task without taking care of what object the robot is going to grasp, the whole point cloud, of tabletop objects, could be giving as input to the service.

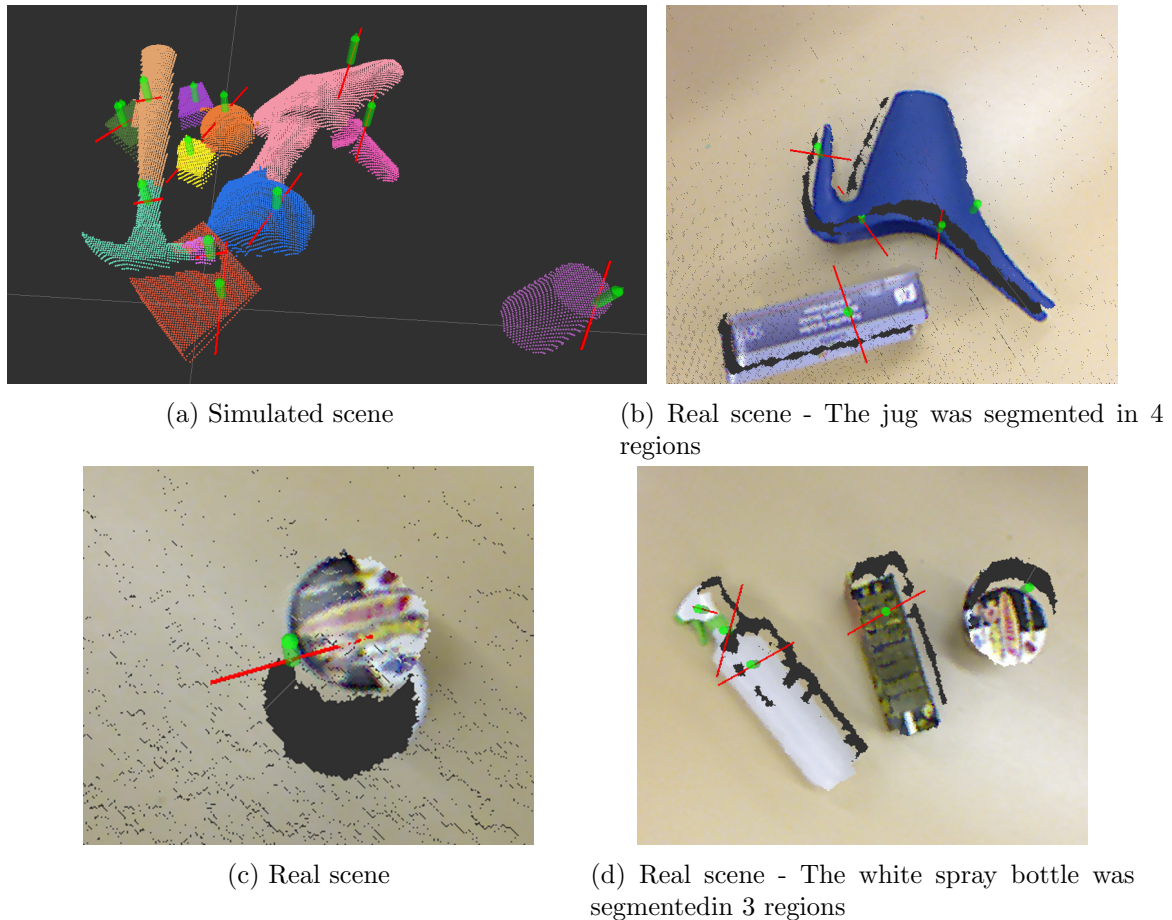


Figure 13: HAF algorithm tests. The green markers show the approaching direction and the red markers show the gripper closing direction. The width closing direction scaled by 2 (Two times bigger).

Analysis In Figure 13 some results are reported. In Figure 13a the results on the simulated scene are shown, the majority of them are good ones. There some unfeasible grasps due to the fact that the objects are considered individually.

An interesting point is that for big objects, for example the red box below the hammer, the algorithm detects a grasp in a corner. This is because the objects is too big for the gripper so it tries to find a feasible grasp accordingly to its maximum opening width. Unfortunately the authors didn't provide a way to increase the opening width of the gripper, also inspecting the code no related variables have been detected, suggesting that it was indirectly included in the support vector machine. Moreover the algorithm does not take into account if a pose is actually feasible, the pose for the red box is clearly unfeasible except for a gripper with a lot of friction.

The algorithm has a variable named `gripper_width` which is a integer variable and it is used only to test grasps with a opening width smaller than the maximum one (see the package's documentation¹²).

Another drawback observed is that the algorithm was never able to detect feasible grasps for cylinder objects such as cups, it can be observed that the grasp is centred near the border of the object, this was a patterned observed in all tests with cylindrical objects (e.g. 13c and 13d).

With simple objects, ones with rectangular shapes, or cylinders lying down, it presents good

¹²http://wiki.ros.org/haf_grasping

results as shown in Figure 13d. With more complex objects, as the jug in Figure 13b also the grasping poses are all feasible.

An interesting example is the one presented in Figure 14, the handle of the shovel is detected as a separated object but with a feasible grasp, the others segment of the shovel does not present any feasible grasp for the vertical approaching direction.

Another thing to take into account is that the width of the grasps is not very reliable. In all Figures relative the HAF algorithm the length of the closing direction markers, which is equal to the width of the grasp, has been scaled by 2 (two times bigger than the width returned by the algorithm). It can be observed that several times, although it is scaled, the width is not actually a feasible one. To avoid this issue the width parameter should be ignored and open the gripper with its maximum opening width for each grasping attempt.

It has been also tested detecting the best grasp for an object not giving only the segmented cloud by giving all the input cloud (the table with the object), with the centred area with respect to the segmented object. The results are worst, the majority of times the algorithm was not able to find any feasible grasp.

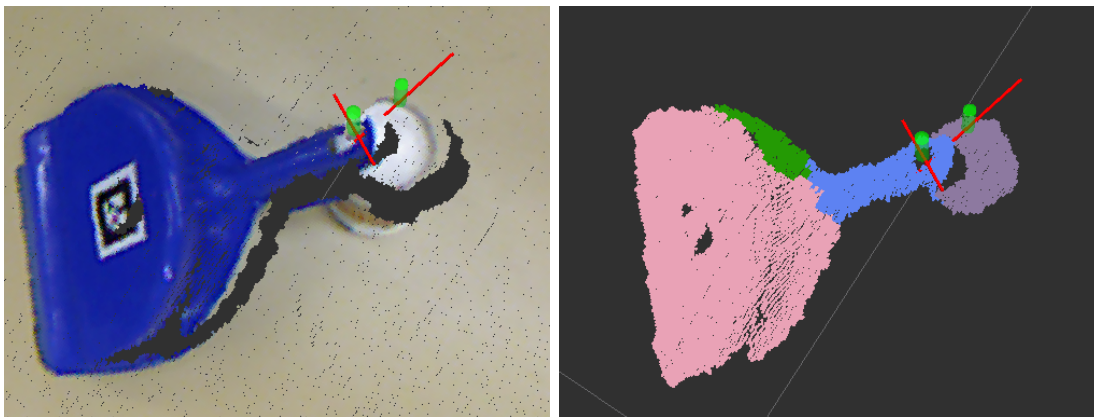


Figure 14: HAF algorithm tests. The green markers show the approaching direction and the red markers show the gripper closing direction. The width closing direction scaled by 2 (Two times bigger).

Further Work The client node could be improved giving the possibility to detect the feasible grasps for several approaching directions, although due to the huge computational cost of the algorithm it would be a good idea to apply it only on a single object, instead of detecting all the grasps for all the objects on the table. Then, the best grasp can be chosen by inspecting its value and then checking for collision or not. If a collision occurs let's test the second one and so on. Also setting a threshold would be a good idea in order to save time avoiding the collision checking for likely unfeasible grasps.

5 Conclusions

An useful segmentation algorithm for cluttered scene of tabletop objects has been designed. This segmentation could be used to find the grasping pose for a certain tabletop object. Unfortunately, the results obtained in the experiments of the grasping pose detection algorithms do not match the ones reported by the authors. Despite this, both present nice results, and present different advantages. The AGILE algorithm is able to predict reliably grasping poses that make sense, the problem in the experiments was probably due to the support vector machine, which should be

trained with the Kinect's set up of the laboratory. In the other hand it does not provide any score to select grasps.

The HAF algorithm provides a score to detect the best grasp among the feasible ones, but a strange behaviour was observed with cylindrical objects, things that do not happen with AGILE. Moreover, it is computationally very expensive with respect to AGILE, and the code cannot be adapted to a different gripper width while the AGILE can be.

As final conclusion, we consider that the most promising algorithm is AGILE, but it needs some improvements as a score function, the training of a new classifier or designing an algorithm to fill occluded sides to work with the set up in the laboratory. In the other hand, the HAF algorithm looks less promising but it could be directly, without any further work, used to manipulate also complex objects, taking care to avoid objects or situations as the one in Figure 13c.

References

- [1] David Fischinger, Astrid Weiss, and Markus Vincze. Learning grasps with topographic features. *I. J. Robotic Res.*, 34(9):1167–1194, 2015.
- [2] Jeremie Papon, Alexey Abramov, Markus Schoeler, and Florentin Wörgötter. Voxel cloud connectivity segmentation - supervoxels for point clouds. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, Portland, Oregon, June 22-27 2013.
- [3] S. C. Stein, M. Schoeler, J. Papon, and F. Woergoetter. Object partitioning using local convexity. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2014*.
- [4] Andreas ten Pas and Robert Platt. Using geometry to detect grasp poses in 3d point clouds. In *International Symposium on Robotics Research (ISRR)*, September 2015.

IRI reports

This report is in the series of IRI technical reports.

All IRI technical reports are available for download at the IRI website

<http://www.iri.upc.edu>.