Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 93 (2020) 508–513

www.elsevier.com/locate/procedia

53rd CIRP Conference on Manufacturing Systems

Object localization utilizing 3D point cloud clustering approach

Gergely Horváth[a,b,*], Gábor Erdős[a,b]

[a] *SZTAKI: Institute for Computer Science and Control, Budapest 1111, Hungary*

[b] *Department of Manufacturing Science and Engineering,Budapest University of Technology and Economics, Budapest 1111, Hungary*

Abstract

In this paper we present a novel method to recognize multiple workpieces on a flat surface, based on a 3D point cloud and a prebuilt pose database. First, the method identifies the main surface, which is the base of every following steps. The surface determines the plane stretched by X and Y axis, establishing the orientation of the objects on it and helping their isolated recognition. Isolation of objects is followed by determining their specific type and exact pose, based on the prebuilt database. The presented method is demonstrated by recognizing workpieces and tool in an assembly cell.

Keywords: identification; object localization; point cloud.

1. Introduction

Robotic assistance is getting prevalent in many fields—e.g. manufacturing, medicine, home assistance or even disaster recovery—because of various capabilities they have. Being resistant toward repetition exhaustion, chemical effects and other environmental hazards, or higher possible precision during manoeuvering is only a fraction of all the leverage a robot may offer. To be able to amply exploit the potential of robots, automatization has to be introduced. This way robotic devices can make self-governing decisions based on sensor inputs, how certain tasks are to be executed. One such task can be object grabbing, usually called pick-and-place in manufacturing.

In the past decade point cloud registering devices, such as Time-of-flight (ToF) cameras, matured to a point, where low-cost devices are already able to capture decent images. As this kind of technology improves, it becomes more and more accessible to capture huge point clouds. The problem to be tackled is associated with the size of the point cloud. Algorithms are created to deal with such point clouds by the authors, see [1], based on these methods a computational library is under development. The potential of these algorithms have already been tested and shown in previous works, demonstrating recognition of complex engineering objects (although built up of mostly cylinders and pipes, detailed in [1]), and robot cell calibration based on previously known artifacts (see [4]). The aim of this article is to make another use-case scenario in which another possible employment of said library is inspected.

In this article a method is presented to localize objects on a plane and highlight those objects on a color image based on a depth image. The color and depth images are assumed to be synchronized, meaning their viewpoint, view direction, vertical vector, and field-of-view are the same. It is also assumed that the objects are on a plane surface and they are not touching each other.

Section 2 presents the background on object localization and recognition. Section 3 demonstrate and detail the method solving the stated problem. Section 4 shows the tests that were carried out. The paper is closed with a section of Conclusion and References.

2. Background on object localization and recognition

Object localization is a commonly researched topic. The idea is to have a suitable representation for an object that can be identified on the image. According to [6, 7] the various definitions defer in the representation of the object location, which

* Corresponding author. Tel.: +36 1 279 6181
E-mail address: gergely.horvath@sztaki.hu (Gergely Horváth).

10.1016/j.procir.2020.04.132

can be either a set of coordinate pairs representing its center point, it can be the bounding box, the object countur, or even a pixel level representation. Lampert et al. [6, 7] introduce a branch-and-bound algorithm to find an object on the image by its bounding box. The proposed approach balances between speed and accuracy. The widely used sliding window algorithm is being extended in this paper. Müller et al. [9] also proposes a sliding window algorithm variaton. They utilize the knowledge of the camera intrinsic parameters to enhance the original method. Object proposal generation sees an increase in popularity and often used as the preprocessing step in a lot of vision systems according to Guo et al. [3]. In[3] the authors compare five object proposal generation algorithms and state The reasoning is that, object proposal generation algorithms tends to perform superior to sliding window algorithms and derivatives, as it is not confined by fixed size.

Quite a handful of articles deal with the topic of object or feature recognition. Feature recognition techniques work either on 2D or 3D images.

According to [14, 10] and [8] a rather common instrument, in the field of object and/or feature recognition, is the application of *feature descriptors*. Feature descriptors are parameter vectors describing part or whole of an image. *Global feature descriptors* represent the whole image with a single vector, making it an information-dense characteristic. Being sensitive to occlusion, clutter and other types of image flaws, they usually work well when segmentation has been applied to the image. In such cases one can be sure, that only a single object is being described. *Local feature descriptors* on the other hand calculate parameter vectors for multiple points of an image, based on local neighborhood characteristics. This will make it more resistant to problems concerning the global feature descriptors. Common usage of local feature descriptors is to match them on different images, making it possible to track changes on an image (like moving of objects).

Applying Convolutional Neural Network (CNN) is also common practice. According to [11], even though the method itself has been introduced in the 1980s, its usage got a huge boost when combination with GPGPU occurred. Most usage of CNN requires strong supervision in the form of time consuming human labour, labeling the artifacts on the images. Vora et al. [12] introduced a completely unsupervised object localization algorithm, however their method only works on images containing only a single object. Although the work of Zhang et al. [13] is involved with manufacturing feature recognition in a single object, it shows the possible problems and hardships around CNN based solutions. According to [13], the following flaws hinder manufacturing feature recognition:

1. Generalization is difficult.
2. Lack noise tolerance regarding the initial CAD models.
3. Computationally intensive and inflexible.
4. Only a limited set of features can be used.

The process detailed in the current paper addresses the problem of segmentation by proposing a light source independent, stable background removal. After segmentation, the application of global feature descriptors becomes feasible. The algorithm can be considered as an object proposal generation algorithm. Although numerous options are available—a handful of them available in [10]—at the end of Section 4 sample usage of a suggested descriptor is layed out. This can be beneficial in scenarios when a high resolution camera is paired with a lower resolution, probably lower cost, depth camera. In this case the depth camera can be used to segment the space, give a hint so to speak, for the object recognition and grasp planning algorithm working with the image of the high resolution camera.

3. Point cloud processing

The method presented in this article, resolving the problem stated in Section 1 is composed of the following steps:

1. segmenting point cloud to find objects.
   - Detect base plane on which the objects are laying.
   - Dropping the points of the base plane.
   - Voxelizing the point cloud.
   - Filtering out sparse voxels.
   - Building the voxel connectivity graph. Connected subsets represent distinct objects.
2. Filtering out small connected subsets of the voxel connectivity graph as irrelevant objects or clutter. The leftover subsets are the relevant objects in space.
3. Visualize the located objects on the color image.
   - Map the object points to the color image.
   - Finding bounding box of the objects, on the color image.

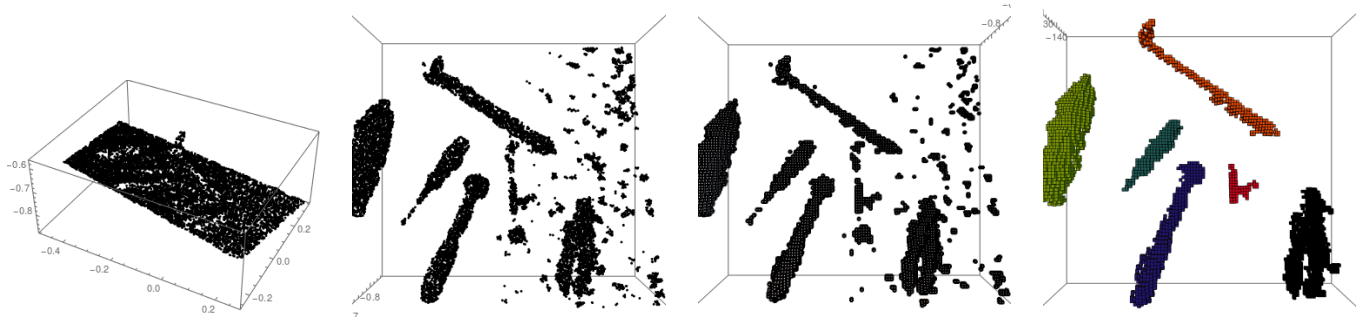
3.1. Voxelization and voxel graph construction

The terminology *voxel* is similar to *pixel* i.e. pixel means *picture element*, while voxel means *volume element*. Point clouds may consist of millions of points, which may lead to lengthy calculations. To condense the data, voxel representation is used. Preservation of the necessary accuracy is ensured by the voxel size, which is chosen such that measurement errors or other technical uncertainties surpass it. It can be said, that the voxel size is the abstraction of the measurement accuracy.

Voxelization is the process of calculating the containing voxels for every point in a given point cloud. Voxels are cubes in the space. The voxels are congruent, axis-aligned and the edge size is a parameter of the voxelizer algorithm. When covering the whole space, there is a voxel which has a vertex in the origin. The faces of the neighboring voxels are touching.

While points are denoted by their spatial coordinates, voxels are represented by *voxel coordinates*. Denote the point coordinates with $p_i = \{x_i, y_i, z_i\}$ and the corresponding voxel coordinates as $v_{p_i} = \{vx_i, vy_i, vz_i\}$, where $p_i \in \mathbb{R}^3$ while $v_{p_i} \in \mathbb{Z}^3$.

For every point the containing voxel is calculated. This essentially is a division, and than only the integer part of the result



(a) Original point cloud (b) Cutting base plane (c) Voxelization with filtering (d) Different objects as color coded VCG subsets

Fig. 1: Result of the different phases of the utilized algorithm

is considered. With this method, the p_i coordinates of the points are converted to v_{p_i} voxel coordinates.

The next step—after finding the mapping between the points and voxels—is to create a voxel connectivity graph. The vertices of this graph are the voxels, while the edges are based on voxel neighborhood. There is an edge between two graph vertices if the two corresponding voxels are connected, either by common face, edge or vertex. In this graph—call it the *voxel connectivity graph*, or *VCG* for short—connected subsets are searched. These subsets will represent the point cloud of the distinct objects in space as shown on Figure 2 and Figure 1.

3.2. Filtering

Now there are a couple of difficulties, that hinder the space partitioning such as:

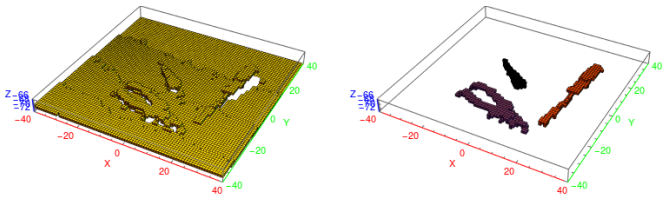
- measurement errors, like reflection,
- dense point cloud, which makes all the voxels connected,
- objects in the space that shouldn't be considered.

To tackle these obstacles, filtering is implemented on multiple levels of the algorithm:

Filtering points to "cut" the point cloud into multiple valuable clouds. In the chosen use-case the objects are positioned on a flat surface. This means that the objects and their point clouds are connected by the points on the surface. This makes the separation of objects difficult. To balance this effect, the surface points are removed (see Figure 2). If a different use-cases were to be considered, it could also be used to remove irrelevant points that are out of the observed space, or based on other criteria.

Filtering voxels with less than a predefined number of points. Although the whole space can be covered with voxels, during calculations only those will be considered which contain a minimum amount of points. Usually there are multiple points in a single voxel. To differentiate between sparsely and densely populated voxels, those that have less then a predefined number of points are filtered. This helps to keep the consistency of the point cloud and filter out random measurement errors.

Filtering VCG subsets with less than a predefined number of voxels. When VCG is created, there still can be objects (which are the connected subsets of the graph) that are small, which usually means being insignificant for engineering use-cases. These objects can also be filtered, being removed from space.



(a) Voxelized point cloud (b) Remaining voxels after plane removal

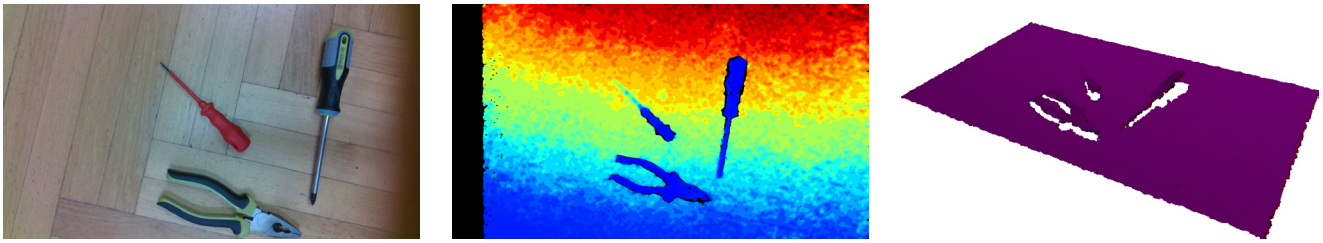
Fig. 2: Usage of initial point filtering. Color indicates connected voxel subsets

It is important to understand, that these filters are always situational and extremely sensitive to their parameters. First of all the size of the point cloud, and the granularity of the point cloud is a factor to be taken into consideration, when deciding on the applied filters and their parameters. It is also an important factor, how aggressive the filters need to be e.g. dropping only slight measurement errors or throwing away whole, unimportant objects. It is also worth mentioning that certain goals, like removing a certain clutter from the initial dataset, may require the symbiotic work of multiple filters.

4. Test scenario

As stated in Section 1 our algorithm is evaluated during a proof-of-concept test case. The goal of these tests were to take an image with color and depth data, find the objects on the image using the depth data and mark them on the colored image.

Multiple sample scenes are considered. Samples consist of a wide variety of objects from industrial tools (screwdrivers,



(a) Color image (b) Depth image (c) Point cloud

Fig. 3: Representations of the same scene, with out-of-sync color and depth images

wrenches, pliers etc.) to everyday objects, such as cups, glasses, boxes, phones etc. These sample objects are layed out on a flat surface, usually a floor or a tabletop. Scenes are captured using an Intel® Realsense™ D435 depth camera, using a custom built software exploiting the `librealsense` library. The library can be accessed through its Github page at [5].

The software initializes the sensor to the appropriate resolution (1280×720 in our case) and executes some graphical setting up. The main (and only) screen of the application constantly updates and presents the color image grabbed from the sensor, which helps finding the proper angle for capturing. After a suitable scene is found, pressing any key captures the images in all the available presets. The library have multiple predefined option kits, called presets, available, however only the ones created with `HIGH ACCURACY` preset are used. Color and depth images are saved, as well as the point cloud, using the `.ply` format and a metadata containing important data about the scene and capture. These data included resolution, intrinsic camera parameters and distortion model—which was Inverse Brown-Conrady Distortion in given test scenarios—among other parameters.

The sensor and the software creates the 3D point cloud data indirectly from the depth image, which can be used to localize the important objects of the scene by the method layed out in Section 3. After the objects are found, their points can be transformed back to the depth image. The conversion between a pixel $p = \{p_x, p_y, d\}$ on the depth image—where $\{p_x, p_y\}$ are the pixel coordinates and d is the corresponding depth value—and its corresponding point P of the point cloud (from which the `.ply` file is generated), looks like $P = \{xd, yd, d\}$, where

$$x = \frac{p_x - PP_x}{F_x} \tag{1}$$

$$y = \frac{p_y - PP_y}{F_y}. \tag{2}$$

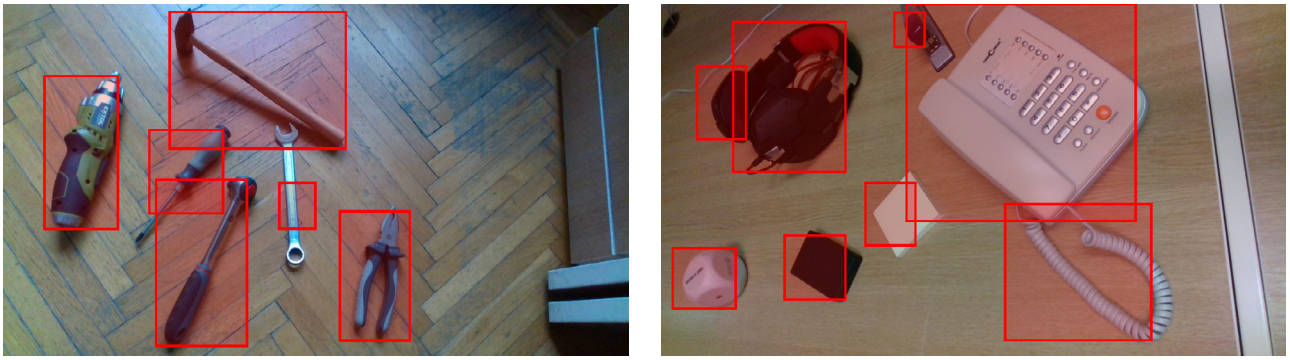
In Equation 1 and Equation 2 PP_x, PP_Y, F_x, F_y are intrinsic camera parameters, extracted from the camera during capturing. This conversion is based on the used distortion model of the camera. It is based on the function `rs2_deproject_pixel_to_point` in the file `librealsense/rsutil.h` at [5]. On Figure 3, the second image shows the captured depth image, where d depth data is color coded (blue is closer, red is farther from the objective). On the same figure the point cloud representation can also be seen, as points in the 3D space.

However, color and depth images are not necessarily in sync. Observing the color and depth images on Figure 3, the effect is quite visible. The reason behind this, is an offset between the physical location of the actual sensors in the camera. This offset is the actual reason, the prebuilt Realsense™-Viewer software cannot be used, to capture the expected images. However, this positional offset can be compensated and the depth and color image frames aligned, applying a number of library functions. The wiki of [5] comprehend the essence of the frame alignment implementation. When the object localization on the point cloud is executed, and the transformation of found object points to the depth image is done, the alignment between the depth and color images is the key to find the same objects on the color image as well.

Now the algorithm detailed in Section 3 is applied onto the point cloud derived from the depth image. First the surface points needs to be removed from the point cloud. Recognizing the main surface of the scene is the first step achieving said goal. This is done by applying a least square plane fitting on the preprocessed point cloud. Preprocessing is a voxelization with only voxel filtering. Every voxel with less then 2 points in it are removed. Although this may take away information from the dataset, it also greatly reduces the noise of it. When voxelization is done, the voxel coordinate of the voxels are considered and viewed as a temporary point cloud. Now on this temporary point cloud a least square, plane fitting is applied. Using the plane and its normal vector, a cutting can be performed as it can be seen on Figure 2.

The leftover point cloud is now voxelized and the voxel connectivity graph is built. From this graph, those subsets—and the corresponding voxels—are filtered that have less than a predefined number of voxels. This number can be set based on experience of earlier problems, on knowledge about the size of the dataset and the voxel size. The result of these operations is a number of disconnected voxel subsets, that individually represent a single object in space.

Now that a representation of the objects is accessible, it has to be visualized on the colored images. To that end, for a given object, all the points in every voxel of the corresponding voxel connectivity subset is taken. For all these points, utilizing first



(a) Industrial tools

(b) Everyday items in an office

Fig. 4: Found objects on the color image

the mapping between the point cloud and the depth image and then the alignment between the depth image and color image, the coordinate of the matching color point is calculated. From these point coordinates the maximum and minimum value for all the coordinates are taken and treated as the coordinates of bounding rectangle, resulting in an axis aligned bounding box as seen on Figure 4.

Another possibility is utilizing a predefined dataset of the stable poses of equilibrium. This database contains the stable equilibrium poses of every object in the scene. These poses are calculated as if objects were lying on a plane. Calculation of these poses can be seen in [2]. Minimal, base plane touching bounding boxes are fit onto the objects in stable poses. Such bounding boxes can also be fitted onto the found objects in the 3D point cloud. Based on the size of the bounding box of the found objects, and the predefined poses, a matching can be created, using the size of the bounding box as a local reference frame based feature descriptor. This matching can be used to label the objects of the scene.

5. Conclusion and future work

Currently the normal vector of the base plane of the test scenario is determined utilizing least square plane fitting. Because of the presence of other objects in the point cloud—those that need to be identified—the least square method will not give a perfect result. Cutting of the point cloud, based on the normal vector of the plane and the set cutting depth, may lead to situations where points of the plane are still preserved, even though objects are removed. Applying a better plane finding (such as RanSaC) algorithm could decrease such effects.

Another field where improvement is possible and necessary is the parameter tuning, which have been done manually during the detailed test scenario. The first parameter that had to be calibrated is connected to the already mentioned plane cutting, the cutting depth. This value can compensate for the errors caused by the flawed plane fitting.

Voxel filter parameter (the minimum number of points, that should be in every voxel), in conjunction with the VCG subset filter parameters, can compensate for a lot of errors. Their relation is a key factor for a usable VCG creation. Although the voxel size represents the intended precision, it can also help connecting or separating voxel subsets, in the voxel connectivity graph. Morphological operations, i.e. dilation, erosion, opening or closing, may also help a lot with partitioning the VCG.

In the article, during the presentation of the test scene there are no run times measured. The reason behind is two fold: first the parameter tuning is not ready and only hand tuning is possible. algorithmic setting of such parameters will introduce the possibility of accurate measure. Second, although it wasn't measured exactly, it was only a couple of minutes (under 5). However, during earlier usage of the presented algorithm, such parameter tunings took tens of minutes. This means that experience with the algorithm has a huge effect on the setup time.

Usually these parameter tunings are corresponding to a certain scene setup, so not moving the camera relative to the surface will most certainly allow the reuse of the hand tuned parameters with favorable results. Parameter reuse is possible when the camera is kept static, however there are no experimental measurements showing what are the errors introduced by the relocation of the sensor.

Another possible utilization aspect of the method is pairing it with machine learning algorithms for image recognition. The initial input of these algorithms are a collection of preprocessed images where objects of interest are marked. As it is shown throughout this paper, the method presented is capable of outputting such images.

6. Acknowledgements

This research has been supported by the GINOP-2.3.2-15-2016-00002 grant on an "Industry 4.0 research and innovation center of excellence". This research has been supported by the ED_18-2-2018-0006 grant on an "Research on prime exploitation of the potential provided by the industrial digitalisation".

References

[1] Erdős, G., Nakano, T., Horváth, G., Nonaka, Y., Váncza, J., 2015. Recognition of complex engineering objects from large-scale point clouds. CIRP Annals - Manufacturing Technology 64, 165 – 168. doi:http://dx.doi.org/10.1016/j.cirp.2015.04.026.

[2] Fekula, M., Horváth, G., 2019. Determining stable equilibria of spatial objects and validating the results with drop simulation. Procedia CIRP 81, 316 – 321. doi:https://doi.org/10.1016/j.procir.2019.03.055. 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, June 12-14, 2019.

[3] Guo, W., Lin, R., Wang, S., Xiong, N., 2018. Object proposal generation for unsupervised object localization, in: 2018 9th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), p. 235–242. doi:10.1109/PAAP.2018.00046.

[4] Horváth, G., Erdős, G., 2017. Point cloud based robot cell calibration. CIRP Annals 66, 145 – 148. doi:https://doi.org/10.1016/j.cirp.2017.04.044.

[5] Intel®, 2015-2020. Intel® Realsense™ sdk. URL: https://github.com/IntelRealSense/librealsense. accessed: 2020-01-28.

[6] Lampert, C.H., Blaschko, M.B., Hofmann, T., 2008. Beyond sliding windows: Object localization by efficient subwindow search, in: 2008 IEEE Conference on Computer Vision and Pattern Recognition, p. 1–8. doi:10.1109/CVPR.2008.4587586.

[7] Lampert, C.H., Blaschko, M.B., Hofmann, T., 2009. Efficient subwindow search: A branch and bound framework for object localization. IEEE Transactions on Pattern Analysis and Machine Intelligence 31, 2129–2142. doi:10.1109/TPAMI.2009.144.

[8] Lisin, D., Mattar, M., Blaschko, M., Learned-Miller, E., Benfield, M., 2005. Combining Local and Global Image Features for Object Class Recognition, in: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops, pp. 47–47. doi:10.1109/CVPR.2005.433.

[9] Müller, J., Fregin, A., Dietmayer, K., 2018. Disparity sliding window: Object proposals from disparity images, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), p. 5777–5784. doi:10.1109/IROS.2018.8593390.

[10] Quan, S., Ma, J., Hu, F., Fang, B., Ma, T., 2018. Local voxelized structure for 3d binary feature representation and robust registration of point clouds from low-cost sensors. Information Sciences 444, 153 – 171. doi:https://doi.org/10.1016/j.ins.2018.02.070.

[11] Simard, P.Y., Steinkrau, D., Buck, I., 2005. Using gpus for machine learning algorithms, in: Proceedings. Eighth International Conference on Document Analysis and Recognition, IEEE Computer Society, Los Alamitos, CA, USA. pp. 1115–1119. doi:10.1109/ICDAR.2005.251.

[12] Vora, A., Raman, S., 2018. Iterative spectral clustering for unsupervised object localization. Pattern Recognition Letters 106, 27 – 32. doi:https://doi.org/10.1016/j.patrec.2018.02.012.

[13] Zhang, Z., Jaiswal, P., Rai, R., 2018. Featurenet: Machining feature recognition based on 3d convolution neural network. Computer-Aided Design 101, 12 – 22. doi:https://doi.org/10.1016/j.cad.2018.03.006.

[14] Zhou, W., Ma, C., Liao, S., Shi, J., Yao, T., Chang, P., Kuijper, A., 2018. Feature fusion information statistics for feature matching in cluttered scenes. Computers & Graphics 77, 50 – 64. doi:https://doi.org/10.1016/j.cag.2018.09.012.