



## labelCloud: A Lightweight Labeling Tool for Domain-Agnostic 3D Object Detection in Point Clouds

Christoph Sager<sup>1</sup> , Patrick Zschech<sup>2</sup> , Niklas Kühl<sup>3</sup> 

<sup>1</sup>Technische Universität Dresden, [christoph.sager@gmail.com](mailto:christoph.sager@gmail.com)

<sup>2</sup>Friedrich-Alexander-Universität Erlangen-Nürnberg, [patrick.zschech@fau.de](mailto:patrick.zschech@fau.de)

<sup>3</sup>Karlsruhe Institute of Technology, [niklas.kuehl@kit.edu](mailto:niklas.kuehl@kit.edu)

Corresponding author: Christoph Sager, [christoph.sager@gmail.com](mailto:christoph.sager@gmail.com)

**Abstract.** The rapid development of 3D sensors and object detection methods based on 3D point clouds has led to increasing demand for labeling tools that provide suitable training data. However, existing labeling tools mostly focus on a single use case and generate bounding boxes only indirectly from a selection of points, which often impairs the label quality. Therefore, this work describes labelCloud, a generic point cloud labeling tool that can process all common file formats and provides 3D bounding boxes in multiple label formats. labelCloud offers two labeling methods that let users draw rotated bounding boxes directly inside the point cloud. Compared to a labeling tool based on indirect labeling, labelCloud could significantly increase the label precision while slightly reducing the labeling time. Due to its modular architecture, researchers and practitioners can adapt the software to their individual needs. With labelCloud, we contribute to enabling convenient 3D vision research in novel application domains.

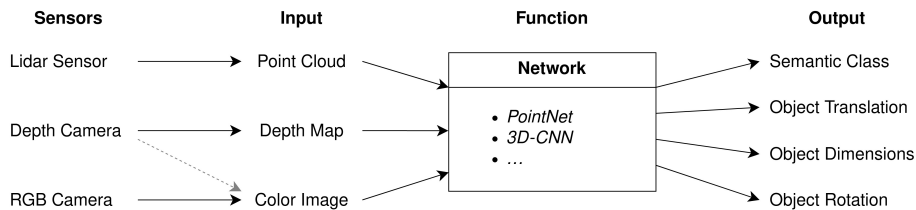
**Keywords:** 3D object detection, labeling tool, point clouds, bounding boxes

**DOI:** <https://doi.org/10.14733/cadaps.2022.1191-1206>

## 1 INTRODUCTION

Within the past decade, the technological advancements in artificial intelligence (AI) in general and machine learning (ML) in specific [11, 12] have led to many significant contributions in the field of image processing and computer vision. Exemplary applications range from face detection and mask recognition [13] over medical diagnoses [14] up to automated quality control in manufacturing [24, 31]. However, new use cases like autonomous driving and collaborative robotics additionally require reliable estimates of object distances and sizes, which has been a problem for traditional ML methods that were based solely on 2D images [9]. This development was accompanied by improving 3D sensors and led to increased interest in the automated analysis of 3D image data [6]. As a result, new ML frameworks are able to process large 3D datasets [17] and can automatically detect objects in them (see Figure 1 for an abstraction of the whole object detection process).

Nevertheless, the challenge remains that almost all solutions in the realm of image recognition are based on some kind of supervised learning and therefore require large amounts of annotated training data [21]. Depending on the final employment—especially for safety-critical systems—it is of utmost importance that the labeling is as accurate as possible to ensure high-quality outcomes of the resulting ML models. Labeling in the 3D space is mostly performed manually by expert workers that draw 3D bounding boxes around target objects. These are used to train ML models, which should later automatically identify the objects of interest, e.g., pedestrians for autonomous driving or tumors within radiography.



**Figure 1:** The 3D object detection pipeline from sensor type to output parameters.

While there exists already a large number of tools for labeling 2D images [21, 11], the selection of tools that support the annotation of 3D data is still limited. The small range of recent non-commercial 3D labeling tools that are freely available all share three major shortcomings: (i) they are specified for autonomous driving applications, (ii) they lack convenience and comfort functions, and (iii) they have many dependencies and little flexibility in data format. Therefore, we propose a novel labeling tool for 3D object detection in point clouds to address these shortcomings. *labelCloud* presents a domain-independent and straightforward approach for creating 3D training data that enables research in 3D vision for new application areas.

## 2 FUNDAMENTALS AND PREVIOUS APPROACHES

The recent progress in autonomous driving was enabled by two major developments: advances in ML and improved sensors. Especially 3D sensors like light detection and ranging (LiDAR) and depth cameras have become more precise and affordable over the years. These devices create 3D reconstructions of their environment by either measuring the time of flight from laser rays (i.e., LiDAR) or comparing the images from parallel cameras (i.e., stereovision). With this technology, it has become easier to determine the distance to and size of surrounding objects, as the parameters do not have to be guessed from two-dimensional images anymore. Moreover, the ubiquity of 3D sensors has made this technology interesting for multiple domains (e.g., robotics, medicine, virtual reality). With the integration of LiDAR into latest smartphones, such as the iPhone 12 Pro, the technology is now spreading into millions of devices, which has sparked interest in the automatic processing and understanding of the data generated. Most 3D sensors output point clouds, i.e., unordered sets of points in Euclidean space:

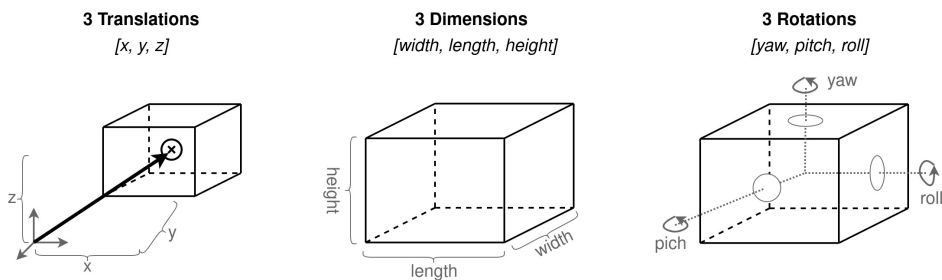
$$\textit{Point Cloud} : P = \{p_1, \dots, p_i, \dots, p_n\}$$

$$\textit{Uncolored Point} : p_i = (x_i, y_i, z_i) \quad x, y, z \in \mathbb{R}^3$$

$$\textit{Colored Point} : p_i = (x_i, y_i, z_i, r_i, g_i, b_i) \quad x, y, z \in \mathbb{R}^3; r, g, b \in [0, 255]$$

The unordered nature of this data type and the absence of any fixed grid (like in 2D images) has made it difficult to simply lift successful solutions from 2D computer vision into the 3D space. However, a new architecture called PointNet [19] has led to a breakthrough and enabled multiple solutions that can detect

objects directly inside the point cloud data [6, 17, 18]. 3D object detection methods can automatically identify and locate objects with their class, position, dimension, and sometimes even rotation (see Figure 2). Still, most existing approaches are based on supervised ML, where an artificial neural network (ANN) is trained for a specific problem. Inspired by information processing in biological brains, ANNs consist of multiple interconnected neurons that are usually structured in stacked layers [11]. The network is trained by applying examples (point clouds) with known outcomes (i.e., objects with their parameters) and subsequently optimizing the edge weights based on the difference between prediction and ground truth. This process is also known as backpropagation and gets iteratively repeated until the improvement converges (for an in-depth introduction of ANNs see [23]). Consequently, researchers and practitioners first have to label large amounts of training data to create accurate ML models.







**Figure 2:** The nine spatial parameters of a 3D bounding box.

Existing non-commercial point cloud labeling tools focus exclusively on the domain of autonomous driving [2, 26, 30]. Thus, they only support data formats and object types that commonly occur in this specific domain. Furthermore, as vehicles tend to be parallel to the floor, most labeling tools consider only the object rotation around the vertical axis (e.g., the heading angle of a car). In doing so, they reduce the dimensionality to a 2D selection from a top-down perspective and infer the bounding box height from the selected points, thus generating the 3D bounding box only indirectly. Direct labeling on the other hand lets the user directly draw a 3D bounding box inside the point cloud. This approach creates some challenges for the user interaction as point clouds are usually very sparse. We developed labelCloud, a lightweight and domain-independent labeling tool for annotating rotated bounding boxes in 3D point clouds. labelCloud supports LiDAR sensors and depth cameras (with seven input formats), multiple label formats ready for use in existing ML frameworks, and the rotation of bounding boxes around all three axes for 6D pose estimation. Table 1 compares our solution with existing approaches in respect to a domain-independent usage.

The major differences are the versatility of the created software in terms of in- and output format as well as usage scenarios. While most existing solutions specialized in a narrow context (fixed import and export formats), we try to foster flexibility and enable a broad usage. Especially the labeling of the x- and y-rotation could be required in domains beside autonomous driving. Also does the support of affordable depth cameras and the easy setup of labelCloud lower the barrier to enter 3D computer vision research in new contexts.

Besides the tools included in the comparison above, platforms providing 3D labeling features commercially are also scarcely available. Nevertheless, these solutions mostly share the limitations of the compared tools and are not accessible to the public. Moreover, the topic of annotating point clouds, in general, was also extensively covered in 2014 at the 9th IEEE Symposium on 3D User Interfaces [1] and led to multiple solutions based on 3D controllers and virtual reality [3, 25]. However, since these approaches require special hardware and do not focus on enabling ML applications, they are not discussed in this work.

		3D Bat [30]	LATTE [26]	SAnE [2]	labelCloud (ours)
Input	*.bin	✗	✓	✓	✓
	*.ply	✗	✗	✗	✓
	*.pcd	✓	✗	✗	✓
	*.xyz	✗	✗	✗	✓
	LiDAR sensors	✓	✓	✓	✓
	Depth cameras	✓	✗	✗	✓
	Colored point clouds	✓	✗	✗	✓
Labeling	Direct	✓	✗	✗	✓
	x-rotation	✗	✗	✗	✓
	y-rotation	✗	✗	✗	✓
	z-rotation	✓	✓	✓	✓
	Orientation	✓	(✓)	(✓)	✓
Setup	Dependencies	8	37	47	4
	No proprietary libraries	✗	✓	✓	✓
	No preparation needed	✗	✗	✓	✓
	No trained models needed	✗	✗	✗	✓
Support	Object tracking	✓	✓	✓	✗
	Point cloud alignment	✗	✗	✗	✓
Code available at					
Fulfilled criteria		8/17	6/17	7/17	16/17

**Table 1:** Comparison of existing point cloud labeling tools for a domain-independent usage.

### 3 SOFTWARE DESIGN

The project was initiated due to the lack of suitable labeling tools for annotating colored point clouds as commonly generated by 3D cameras (like the Intel RealSense series). Researchers wanting to tap into that data faced the problem that existing software was either very complex to set up or incompatible with the desired data formats (like .ply or .pcd). Consequently, we developed a solution that would let users quickly generate annotated training data from their depth sensors to accelerate the research in 3D vision—independent of the respective domain. The design cycle was initiated by a literature review and expert interviews to extract requirements for a suitable solution, which then guided the final software development.

#### 3.1 Requirements Definition

First, we conducted a systematic literature review and three interviews with experts from the industry to collect requirements for a generic point cloud labeling tool. While the literature focused mostly on improvements in labeling time, annotation quality, and ease-of-use, the industry experts also emphasized aspects like learnability and the possibility to further adapt and extend the software to individual applications.

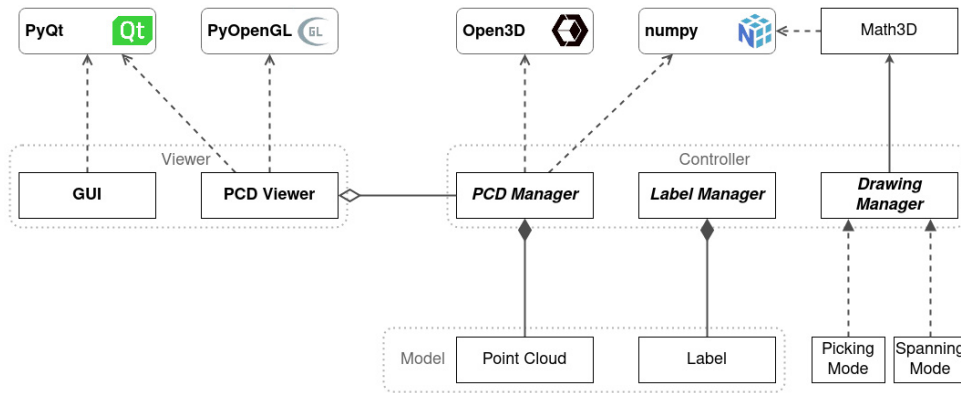
Requirements
R1 Tool must support intuitive and direct labeling inside the point cloud.
R2 Tool must support the annotation of all required parameters of 3D bounding boxes.
R3 User interface should decrease necessary user interactions and labeling time.
R4 Labeling methods should increase label quality and precision.
R5 Tool must have the ability to import point clouds and export labels in multiple formats.
R6 Software stack should allow for fast loading of large point cloud files.
R7 Tool should be configurable so that it can be adapted to the user needs.
R8 Tool should not have too many external dependencies.

**Table 2:** Extracted requirements from literature and expert interviews.

The research led to eight requirements that shaped our software design (see Table 2). The requirements cover the future user base as well as the functionality and mutability of the software. As with 2D labeling, the interviewed experts required the software to be usable also by untrained workers in order to allow an outsourcing of the labeling task. The labeling tool should be compatible with a multitude of available 3D object detection and 6D pose estimation frameworks, thus providing an export of the necessary parameters. Furthermore, as 3D labeling is even more time-consuming than the 2D counterpart, the software should decrease the costs for labeling to a level that allows an economic application of the technology. Another requirement stems from the compute-intensity of processing point clouds. A suitable solution must also run fluently on standard computer hardware and should be designed in a mutable fashion to allow for specialization and extension to various applications. Eventually, the tool should be lightweight with an easy setup so that users can quickly test their projects without having to install heavy dependencies.

#### 3.2 Software Architecture

Based on the defined requirements the software was designed in a modular fashion and developed with the flexible Python programming language. This allowed the integration of external modules like Open3D [29] and NumPy [10], which excel at point cloud processing, respectively array calculations. Overall, labelCloud



**Figure 3:** Architecture of labelCloud with the integration external libraries.

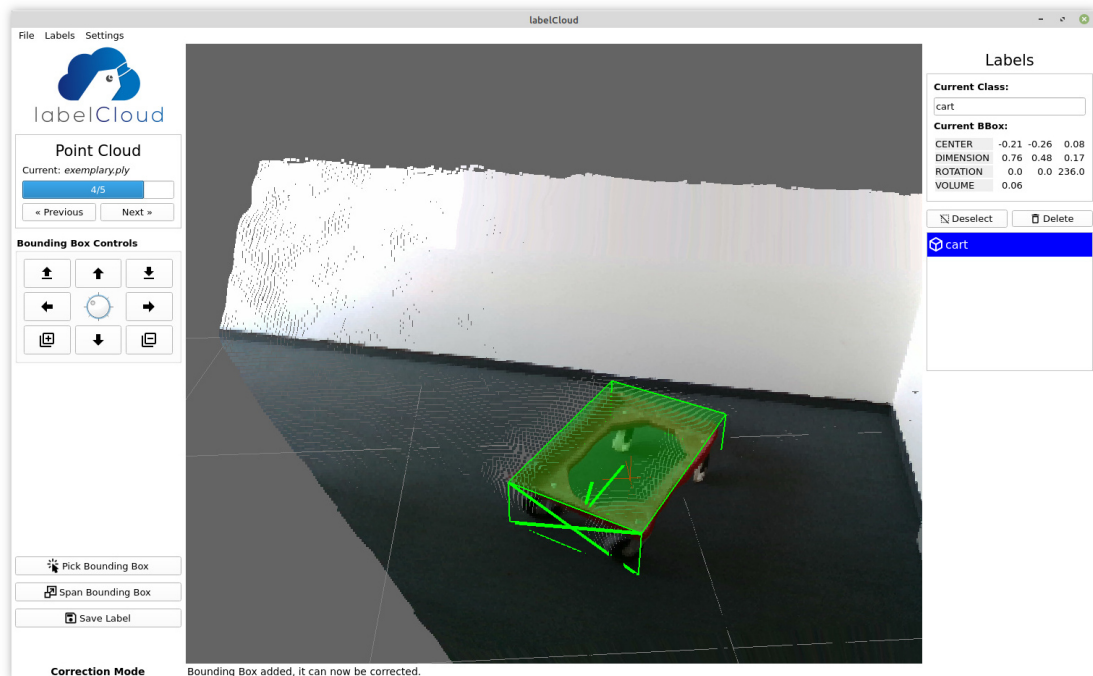
is structured according to the MVC design paradigm [7] into the three main compartments model, view, and control (see Figure 3). The model captures the representation of the point clouds and labels. labelCloud can import seven different point cloud formats from LiDAR sensors (uncolored) as well as depth cameras (colored). We make use of the Open3D library for most formats and implement a custom loader based on NumPy for binary files (see Table 3). Each label is associated with one point cloud and can contain multiple 3D bounding boxes each representing one object. A bounding box consists of up to 10 parameters (see Figure 2 on page 1193): 1 for the object class, 3 for the location ( $x, y, z$ ), 3 for the dimensions (length, width, height), and 3 for the rotations (roll, pitch, yaw). Consequently, users can accurately label objects together with their size and full pose. The created labels can be exported in four different formats including the popular KITTI [8] format, allowing easy integration with existing frameworks.

Import Method	Point Cloud Formats
Open3D	ply, pcd, pts, xyz, xyzn, xyzrgb
NumPy	bin (Velodyne)

**Table 3:** Supported point cloud formats with their import method.

The large size (up to multiple million points) and unordered structure of point clouds make the navigation and interaction a computationally expensive task. Thus, labelCloud's view makes use of the parallel processing power from the GPU and uses OpenGL for a fluent visualization. The point cloud data is transferred at the beginning of each labeling task and any transformation is realized using the projection matrix. This setup allows the fluent annotation of the often large point cloud files with standard computer hardware. The point cloud is rotated (left click) and translated (right click) using prevalent mouse commands. Additionally, the user interface provides buttons and text fields for visual user interaction. Figure 4 shows the GUI of labelCloud after a colored point cloud has been loaded and a first object (i.e., a red transport cart) has been labeled with a 3D bounding box. In this state, the left hand panel shows the manipulation options and the right hand panel gives information about the created labels with the active one being highlighted.

The software behavior is encapsulated in three specialized modules inside the controller compartment. The point cloud manager is responsible for the import, navigation, and manipulation of the point clouds. It keeps track of the progress (share of point clouds labeled) and maps to the underlying file structure. The drawing manager abstracts the different labeling methods (see Section 4.1). It always has one active labeling mode



**Figure 4:** User interface of labelCloud with a labeled point cloud.

to which user interactions are forwarded and is responsible for the live preview of the drawn bounding box. We implement two concrete labeling modes and also provide an abstract template method with a standard interface. This setup allows other developers to add new custom labeling methods and adapt the tool to their needs. The label manager exports the annotated labels (bounding box + class) to a file in the chosen format. We provide three export formats based on JSON and an implementation of the KITTI format. Two of our formats explicitly export all bounding box parameters with an absolute or relative rotation and one format exports all eight vertices of the bounding box. Through its abstract design, labelCloud allows an easy adaptation of existing or extension of additional behavior. With this design, we encourage developers to base more specialized labeling tools on our foundation.

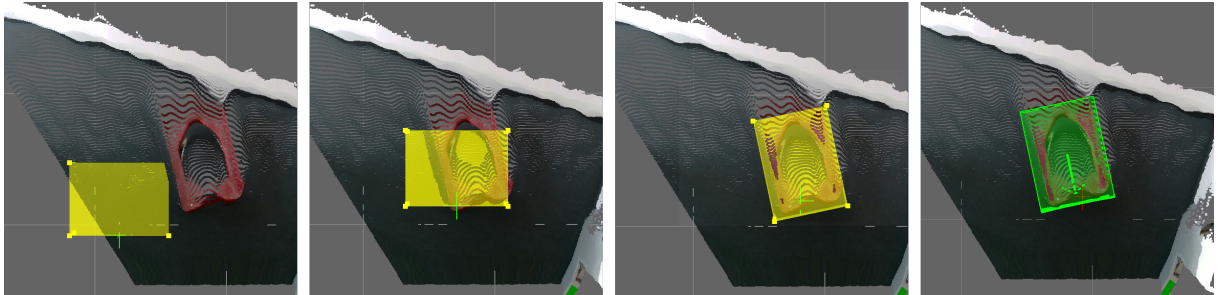
## 4 USER INTERACTION

The labeling process usually consists of three phases: (i) object detection, (ii) bounding box creation, and (iii) parameter correction. Especially in uncolored point clouds, it can take the user a long time to locate and identify an object. Once that has been done, the user has to enter the object class and create an initial bounding box by using one of the provided labeling modes. The 3D interaction inside the point cloud with a 2D controller brings some challenges that we solved with a depth estimation method.

### 4.1 Labeling Modes

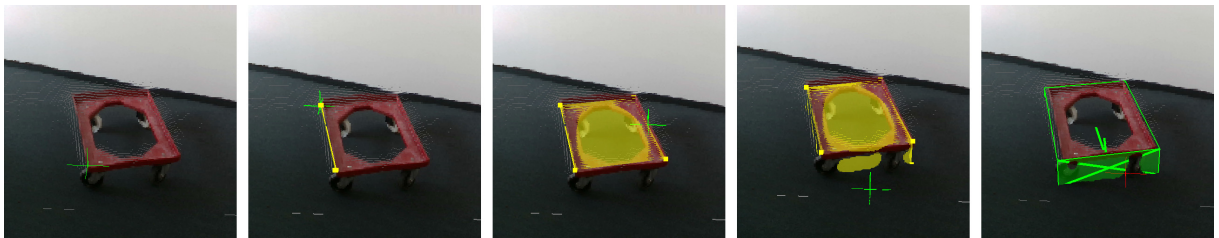
The creation of a bounding box requires the user to specify the location, size, and rotation of an identified object. While for images a 2D bounding box could be spanned using only one to two clicks, this process is much more complex for 3D bounding boxes. A labeling method can either solve all steps at the same time

or only provide an initial draft of the bounding box. We implement the two labeling methods *picking* and *spanning* as well as several possibilities to subsequently improve the parameters of the created bounding box. Furthermore, we introduce a feature for aligning skewed point clouds with the floor. A live demonstration of these features can be seen in our demonstration video (<https://youtu.be/8GF9n1WeR8A?t=126>).



**Figure 5:** Task sequence of the picking mode. The user translates an initial preview of a bounding box (yellow) to the object's location (a & b). The bounding box rotation can be adjusted in parallel using the mouse wheel (c). The final bounding box (green) is specified with a left mouse click (d).

The *picking mode* is based on the assumptions that object sizes are previously known or do not vary too much (e.g., standardized package boxes). It provides a default bounding box with fixed dimensions, which the user can simply drag and rotate into the point cloud (see Figure 5 a-d). As point clouds have a third dimension, the default bounding box automatically adjusts its size if objects are further in the distance. The z-rotation of the bounding box can be adjusted by scrolling the mouse wheel. The x- and y-rotation, which are rarely annotated, can only be adjusted afterward. A yellow shading gives the user a live preview of how the resulting bounding box would look like. Once the location is specified, all other parameters can be freely adjusted. After some testing, we experienced that it is challenging to pick the actual center of an object as it is often occluded by the object surface. Consequently, we now let the user specify the location by picking the front top edge of the object.

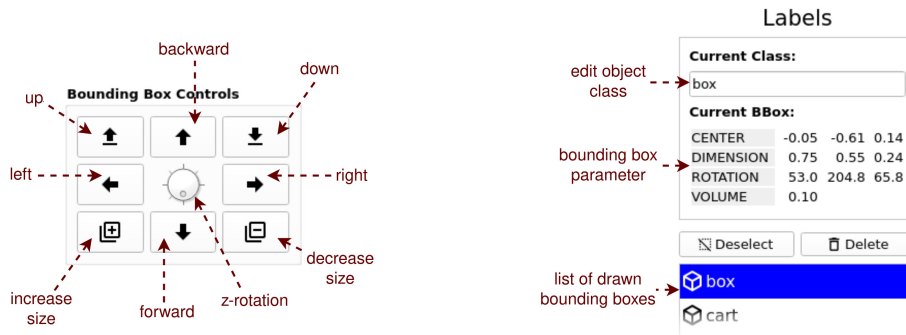


**Figure 6:** Task sequence of the spanning mode. The user selects two vertices representing the length of an object (a & b). Afterwards the width (c) and height (d) of the bounding box get specified with the help of locked dimensions. The final bounding box (green) is calculated on the fourth click (e).

With the *spanning mode*, we tried to lift common 2D labeling methods into the 3D space. Instead of selecting two opposing rectangle corners, the user spans the 3D bounding box using four subsequent clicks (see Figure 6 a-e). Once the user specified the length and rotation with the first two clicks, the selection is supported by locking irrelevant dimensions. The third click is conducted with a locked z-dimension as we can calculate the bounding box width from the perpendicular distance to the previously defined line. The last click



pulls the rectangle into a 3D bounding box and has only one degree of freedom (e.g., z-axis). We calculate the bounding box height from the orthogonal distance to the previously spanned rectangle. This locking approach comes with the advantage that the user does not necessarily have to select points of the actual object but can make use of surrounding elements with the same depth or height. Especially with objects that sit on the floor, the height can be specified simply by selecting a point on the ground. Our evaluation with test users shows that the spanning mode significantly reduces the necessary user interactions by specifying nine parameters using solely four clicks (see Section 5). We also provide a “z-rotation only” mode for use cases, where all bounding boxes should be parallel to the floor (i.e., x-y-plane).



**Figure 7:** Options for bounding box correction (left) and parameter display (right).

Once an initial bounding box has been created, its parameters can be corrected using a selection of key combinations and visual buttons (see Figure 7). Furthermore, labelCloud offers a novel user interaction paradigm called “side pulling”. As it is very labor-intensive to manually define each object dimension, we allow individual changes to the length, width, or height of a bounding box using the mouse. The user just has to hover the cursor over the specific bounding box side and can then use the mouse wheel to push or pull the selected side, thus adjusting its perpendicular dimension. For more fine-grained corrections, the user can control each bounding box parameter with a keyboard shortcut. The granularity of these actions can be adapted in an extensive configuration, thus allowing to adapt the tool to different levels of detail. The right side of the user interface shows the exact parameter values of the current bounding box together with the volume and allows the user to switch to a previously created bounding box. Furthermore, it provides a text field for changing the object class that is supported by an autocomplete feature based on previous labels or a provided list of object classes.

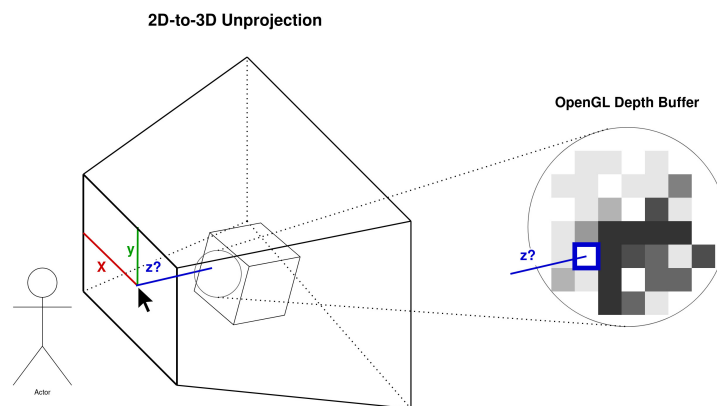
## 4.2 Point Selection and Depth Estimation

The labeling interactions inside the viewer require the selection of specific points from the point cloud. However, three-dimensional selection with only two-dimensional visualization (screen) and input devices (mouse) is a difficult problem as mouse clicks only return information about two dimensions (x and y). We overcome this challenge by estimating the third dimension (z) based on cues and assumptions about the user intent (see Figure 8). Most software solves this problem using ray casting, i.e. taking the depth of the first object that intersects with a perpendicular ray shot from the click coordinates. This approach falls short for point clouds due to sparsity and occlusion, resulting in a volatile selection process. Depth sensors naturally produce sparse point clouds as the scene is recorded from a fixed perspective, but now the user can also look at the scene from other perspectives. Consequently, there is a high chance the ray will not cast any point and go on into infinity

[15]. We approach this problem by not only looking at the clicked pixel and its depth but also considering the surrounding pixels. For this purpose, we make use of OpenGL's depth buffer, which holds a depth value for every screen pixel that corresponds to the closest object to the screen [22]. This value is a float between 0 and 1, where 0 corresponds to the near plane (screen) and 1 represents the far plane (maximum depth) of OpenGL's projection frustum. Furthermore, we introduce two assumptions about the user intent:

1. The user always wants to select a point from the point cloud (i.e., never wants to select infinity).
2. The user is more likely to select the closest point to the screen (i.e., on the object surface).

Based on these assumptions we introduce *depth smoothing* and *depth minimization*. Depth smoothing attempts to solve the sparsity problem if the user fails to directly select a point. Instead of using the maximum allowed depth (far plane), which is certainly not the intent of the user, we average the depth values of surrounding points that the user might have tried to select. This case is detected using a threshold value ( $depth < 0.99$ ) and leads to sending the depth array in a radius of 15 pixels around the click coordinates to a smoothing function. Here, the depth values are filtered for validity (i.e., only of surrounding points) and then the median depth is returned. We have also experimented with the mean as a metric but achieve a more natural selection using the median. The effect of this method can be compared to the snapping feature in other CAD software (like AutoCAD) and leads to a successful selection, even if the user misses the point.



**Figure 8:** Point selection via depth estimation ( $x$  &  $y$  is given;  $z$  is unknown).

Depth minimization, on the other hand, is always employed when the user actually clicks on a point. We noticed that even in this case some post-processing helps to increase the bounding box precision. Oftentimes not the outermost point of an object is hit, which leads to object points not included in the 3D bounding box. For this case, we use the fact that the users most of the time directly face the object of interest and pick the corner that is closest to screen. Especially in the picking mode, the user then has to select a point on the most outer edge of the object. As the users probably choose the edge that is facing them, we assume they want to select the points closest to the screen, thus having a minimum depth value. Consequently, the minimal valid depth value in a radius around the click is used as the  $z$ -coordinate. To reduce surprising effects of this feature, the minimization is done with a smaller radius than the smoothing. Depth minimization is designed to support the users in selecting the right points and reduces the need for subsequent bounding box corrections.

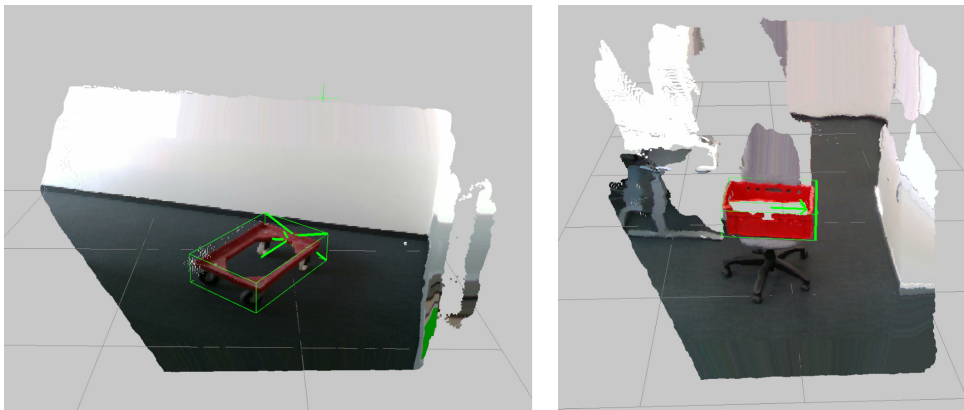
### 4.3 Point Cloud Alignment

Although many depth sensors already come with an inertial measurement unit (which can report the device's orientation), most of them directly export the plain point cloud as faced by the camera sensor. If the sensor direction is not parallel to the floor, this leads to skewed point clouds that are not aligned with the world coordinate system. However, most users want to annotate objects, which are parallel to the floor (e.g., vehicles). labelCloud offers a feature to quickly align a tilted point cloud with a visible floor, so that ground points have a z-value of close to zero. The user only has to select three different points from the floor plane, which are then used to find a transformation matrix that translates and rotates the point cloud from the initial perspective of the sensor to a coordinate system, where the x-y-plane is aligned with the floor.

Other approaches applied automatic alignment based on the heuristic that the lowest points of the initial point cloud already hint to the ground location [26]. However, this technique did not work for us when there were other large objects with smooth surfaces in the point cloud. In one case, for example, the cupboard in an office was wrongly detected as the floor. Consequently, we still require a manual input of the user for the point cloud alignment. Nevertheless, a combination with the automatic technique—reducing the interaction to a single click—might still be rational.

## 5 EVALUATION

The developed software was analyzed in a twofold evaluation. First, the user interaction and labeling modes were tested with three novice and four expert users. The experts were defined by having computer vision or CAD experience in general but not with point cloud labeling in specific. Second, the technical performance of the underlying architecture was stress-tested using large test datasets.



**Figure 9:** Two point clouds from the test dataset with the objects to be labeled (transport carts and box).

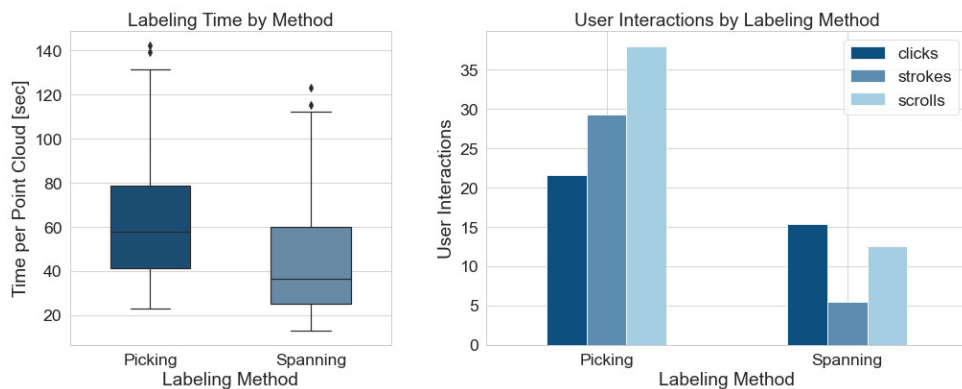
### 5.1 User Evaluation

In a first user evaluation, the participants were asked to label 16 different point clouds using labelCloud. The point clouds were recorded in an office environment with a standard depth camera and contained transport carts and boxes (see Figure 9 and the example point cloud in our GitHub repository [20]). During labeling, the users were free to choose whichever annotation mode and control option they preferred. All user interactions were recorded in the background. The tool and its labeling methods were quantitatively analyzed based on the effort (time and user interactions) it took the users to label all point clouds and the precision of the

drawn bounding boxes as measured with the intersection over union (IoU, see eq. (1)). The IoU measures the overlapping area between the bounding box  $BB$  and the actual ground truth  $GT$  (a gold standard specified by the authors) as a share of the union of both areas [27]. In the case of 3D bounding boxes, the 2D-projections from the top-down perspective are used together with differences in height [28]. The most common object detection competitions use IoU-thresholds of 50% or 25% for a true positive detection with 100% meaning a perfect representation of the object.

$$IoU = \frac{BB \cap GT}{BB + GT - BB \cap GT} \quad (1)$$

On average the test users achieved an average IoU of 67% on the indoor test dataset with rotated objects, taking about a minute to label each point cloud. We did not find any remarkable differences in the performance between expert and novice users. While both of labelCloud's annotation modes led to comparable bounding box precision, the spanning mode required significantly less labeling time (-22%) and user interactions (-63%) compared to the picking mode (see Figure 10). This stems mainly from the fact that the spanning mode already creates a relatively precise initial bounding box, which requires fewer subsequent correction actions. In fact, the users spent only half as long correcting the bounding box when using the spanning mode ( $\varnothing$  17 seconds) in comparison to the picking mode ( $\varnothing$  35 seconds). Also, in a follow-up questionnaire, the users described the spanning mode as the more intuitive and effective approach.

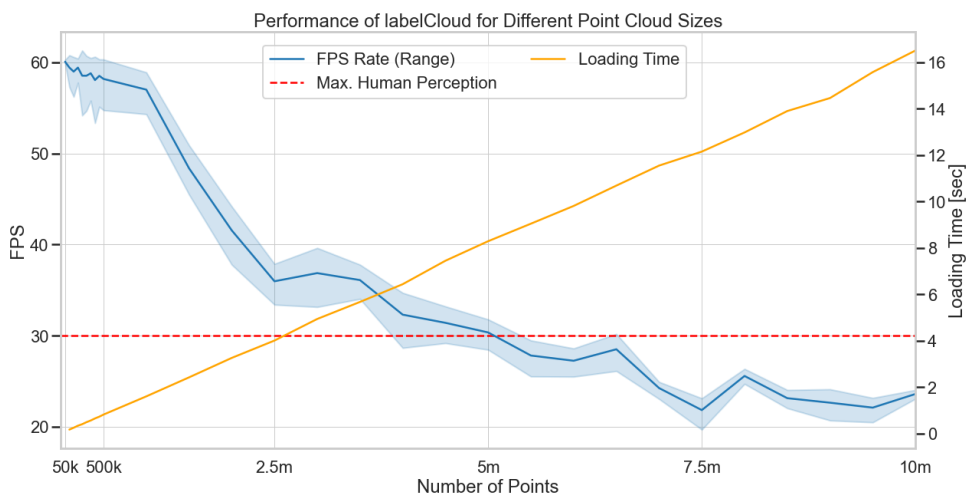


**Figure 10:** Evaluation of the picking and spanning labeling method.

In addition to the evaluation of labelCloud, we also asked the participants to label the same point clouds with a simple labeling tool based on Open3D's visualizer and selection feature that uses an indirect labeling approach [5]. Contrary to spanning a 3D bounding box once, the users iteratively selected the points of an object with a 2D bounding box from multiple perspectives. Thereafter, a script transformed the resulting point subset into a 3D bounding box. While this approach had similar labeling times, it also came with a 30% lower precision (51% IoU) compared to labelCloud. This strengthens our thesis that direct labeling approaches on average lead to more precise bounding boxes, as the user directly draws inside the point cloud and therefore has more control over the final parameter values. Indirect labeling often suffers from outlier points that have not been removed during the selection phase. For example, if a single point from the floor is still selected when the bounding box is generated, this will dramatically increase the resulting bounding box size. In addition, the test users rated the performance of the direct labeling in labelCloud almost twice as good as the indirect labeling approach. Nevertheless, the results are based on a small-scale study and should be validated in a more comprehensive evaluation.

## 5.2 Technical Evaluation

In a second evaluation, the technical aspect of the software was tested. As the software should represent a suitable solution to quickly create training data for various domains, it must be robust in loading and operating large point clouds. labelCloud's setup takes less than five minutes, as it only requires four selected external libraries, and its source is smaller than 1 MB. It can be downloaded and installed using two command-line instructions on all common operating systems. During performance tests on a standard consumer notebook (16GB RAM, mobile GPU), it could seamlessly load and handle point clouds with up to five million points, while still rendering with more than 30 FPS (see Figure 11). Only after this point, the frame rate dropped into an area where the human vision starts to notice lags [16]. However, even with ten million points, the user interface was still usable in a productive way. The time it takes to load the point cloud, on the other hand, is increasing linearly with the size of the point cloud. Still, for the usual point cloud sizes (up to 500k), the loading time stays under a second and is therefore negligible. The results prove that labelCloud is also suitable for domains with extremely large data, like aerial observation [4].



**Figure 11:** Rendering performance of labelCloud for large point clouds.

## 5.3 Limitations

The idea of labelCloud is to avoid a strong bounding to only a single use case (like autonomous driving) and instead to provide a solution with a certain universality in terms of domains and file formats. We hope to provide a first convenient solution to address domain-independent labeling. However, the domain-independence comes with its own drawbacks and thus makes it difficult to directly compare with existing solutions (like in Table 1). A major advantage of specialized labeling tools is that they can adapt to the properties of their domain. In case of LATTE [26] and SAnE [2] this was done with reasonable defaults (like predefined object classes), highly-specialized labeling modes (“one click annotation” with clustering) and pre-trained models. Especially the last two features can speed up the labeling process significantly as the user only has to hint at the rough object location or correct an automatically generated bounding box.

While labelCloud has similar or even better (in case of 3D BAT [30]) precision levels, its labeling times are in part much longer compared to existing tools. The authors of SAnE [2], for example, report average labeling times of down to nine seconds when all automatization features were activated. This is due to the fact

that without any assumptions about the objects to label, it is very difficult to include features that automate at least part of the labeling process. In the end, clustering or pre-trained models always need to optimize either parameters or neural networks towards a specific objective (i.e., object types, sizes, patterns, etc.). We decided to avoid this fixation and instead provide a solution that does not restrict the users to specific domains and data properties. However, we are thinking about extending the software with object tracking that could project bounding boxes – once drawn – into successive frames.

## 6 CONCLUSION

Labeling objects in 3D point clouds is a crucial task to generate training data for ML models in various domains. Existing labeling tools are typically designed for limited settings and do not incorporate, amongst others, aspects of no-frills and direct labeling inside the point cloud. Therefore, we introduce labelCloud which allows for lightweight, convenient 3D point cloud labeling with many different formats without a focus on a specific domain. Our first evaluations show the functionality of the tool with a real-world use case and prove the suitability even for untrained users. Furthermore, does the introduced spanning mode reduce the user effort significantly compared with the baseline. When directly contrasted with an indirect labeling approach, labelCloud increased the bounding box precision by over 30%. In future iterations, we plan to further improve the usability for novice users and introduce object tracking as well as additional labeling modes to decrease the average labeling time per point cloud. A demonstration of the tool can be found on YouTube (<https://www.youtube.com/watch?v=8GF9n1WeR8A>) and the code is available as open-source in our GitHub repository (<https://github.com/ch-sa/labelCloud>).

## ORCID

*Christoph Sager*, <http://orcid.org/0000-0002-7060-0541>

*Patrick Zschech*, <http://orcid.org/0000-0002-1105-8086>

*Niklas Kühn*, <http://orcid.org/0000-0001-6750-0876>

## REFERENCES

- [1] Anatole Lécuyer; Rob Lindeman; Frank Steinicke: IEEE Symposium on 3D User Interfaces (3DUI). In 2014 IEEE Symposium on 3D User Interfaces (3DUI). IEEE, Minneapolis, MN, 2014. ISBN 978-1-4799-3624-3.
- [2] Arief, H.A.; Arief, M.; Zhang, G.; Liu, Z.; Bhat, M.; Indahl, U.G.; Tveite, H.; Zhao, D.: SANe: Smart Annotation and Evaluation Tools for Point Cloud Data. IEEE Access, 8, 131848–131858, 2020. ISSN 2169-3536. <http://doi.org/10.1109/ACCESS.2020.3009914>. Conference Name: IEEE Access.
- [3] Bacim, F.; Nabyouni, M.; Bowman, D.A.: Slice-n-Swipe: A free-hand gesture user interface for 3D point cloud annotation. In 2014 IEEE Symposium on 3D User Interfaces (3DUI), 185–186, 2014. <http://doi.org/10.1109/3DUI.2014.6798882>.
- [4] Boyko, A.; Funkhouser, T.: Cheaper by the dozen: group annotation of 3D data. In Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14, 33–42. ACM Press, Honolulu, Hawaii, USA, 2014. ISBN 978-1-4503-3069-5. <http://doi.org/10.1145/2642918.2647418>.
- [5] Friederich, J.; Sager, C.: ch-sa/threed\_od: An Open3D-based Labeling Script for Rotated 3D Bounding Boxes, 2021. <http://doi.org/10.5281/zenodo.4591310>.
- [6] Friederich, J.; Zschech, P.: Review and Systematization of Solutions for 3D Object Detection. In 15th International Conference on Wirtschaftsinformatik (WI), 1699–1711. GITO Verlag, Potsdam, Germany, 2020. ISBN 978-3-95545-335-0. [http://doi.org/10.30844/wi\\_2020\\_r2-friedrich](http://doi.org/10.30844/wi_2020_r2-friedrich).

- [7] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns: Abstraction and Reuse of Object-Oriented Design. In M. Broy; E. Denert, eds., *Pioneers and Their Contributions to Software Engineering*, 361–388. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 978-3-540-42290-7. [http://doi.org/10.1007/978-3-642-48354-7\\_15](http://doi.org/10.1007/978-3-642-48354-7_15).
- [8] Geiger, A.; Lenz, P.; Stiller, C.; Urtasun, R.: Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, 32(11), 1231–1237, 2013. ISSN 0278-3649, 1741-3176. <http://doi.org/10.1177/0278364913491297>.
- [9] Geiger, A.; Lenz, P.; Urtasun, R.: Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 3354–3361. IEEE, Providence, RI, 2012. ISBN 978-1-4673-1228-8. <http://doi.org/10.1109/CVPR.2012.6248074>.
- [10] Harris, C.R.; Millman, K.J.; van der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.; Kern, R.; Picus, M.; Hoyer, S.; van Kerkwijk, M.H.; Brett, M.; Haldane, A.; del Río, J.F.; Wiebe, M.; Peterson, P.; Gérard-Marchant, P.; Sheppard, K.; Reddy, T.; Weckesser, W.; Abbasi, H.; Gohlke, C.; Oliphant, T.E.: Array programming with NumPy. *Nature*, 585(7825), 357–362, 2020. ISSN 0028-0836, 1476-4687. <http://doi.org/10.1038/s41586-020-2649-2>.
- [11] Janiesch, C.; Zschech, P.; Heinrich, K.: Machine learning and deep learning. *Electronic Markets*, 2021. ISSN 1019-6781, 1422-8890. <http://doi.org/10.1007/s12525-021-00475-2>.
- [12] Jordan, M.I.; Mitchell, T.M.: Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245), 255–260, 2015. ISSN 0036-8075, 1095-9203. <http://doi.org/10.1126/science.aaa8415>.
- [13] Kühl, N.; Martin, D.; Wolff, C.; Volkamer, M.: “Healthy surveillance”: Designing a concept for privacy-preserving mask recognition AI in the age of pandemics, 2021. <http://doi.org/10.24251/HICSS.2021.206>.
- [14] McKinney, S.M.; Sieniek, M.; Godbole, V.; Godwin, J.; Antropova, N.; Ashrafiyan, H.; Back, T.; Chesus, M.; Corrado, G.S.; Darzi, A.; Etemadi, M.; Garcia-Vicente, F.; Gilbert, F.J.; Halling-Brown, M.; Hassabis, D.; Jansen, S.; Karthikesalingam, A.; Kelly, C.J.; King, D.; Ledsam, J.R.; Melnick, D.; Mostofi, H.; Peng, L.; Reicher, J.J.; Romera-Paredes, B.; Sidebottom, R.; Suleyman, M.; Tse, D.; Young, K.C.; De Fauw, J.; Shetty, S.: International evaluation of an AI system for breast cancer screening. *Nature*, 577(7788), 89–94, 2020. ISSN 0028-0836, 1476-4687. <http://doi.org/10.1038/s41586-019-1799-6>.
- [15] Monica, R.; Aleotti, J.; Zillich, M.; Vincze, M.: Multi-label Point Cloud Annotation by Selection of Sparse Control Points. In *2017 International Conference on 3D Vision (3DV)*, 301–308, 2017. <http://doi.org/10.1109/3DV.2017.00042>. ISSN: 2475-7888.
- [16] Nasiri, R.M.; Wang, J.; Rehman, A.; Wang, S.; Wang, Z.: Perceptual quality assessment of high frame rate video. In *2015 IEEE 17th International Workshop on Multimedia Signal Processing (MMSP)*, 1–6. IEEE, Xiamen, China, 2015. ISBN 978-1-4673-7478-1. <http://doi.org/10.1109/MMSP.2015.7340831>.
- [17] Qi, C.R.; Chen, X.; Litany, O.; Guibas, L.J.: ImVoteNet: Boosting 3D Object Detection in Point Clouds With Image Votes. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 4403–4412. IEEE, Seattle, WA, USA, 2020. ISBN 978-1-72817-168-5. <http://doi.org/10.1109/CVPR42600.2020.00446>.
- [18] Qi, C.R.; Litany, O.; He, K.; Guibas, L.: Deep Hough Voting for 3D Object Detection in Point Clouds. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 9276–9285. IEEE, Seoul, Korea (South), 2019. ISBN 978-1-72814-803-8. <http://doi.org/10.1109/ICCV.2019.00937>.
- [19] Qi, C.R.; Su, H.; Mo, K.; Guibas, L.J.: PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 77–85. IEEE, Honolulu, HI, 2017. ISBN 978-1-5386-0457-1. <http://doi.org/10.1109/CVPR.2017.16>.

- [20] Sager, C.: ch-sa/labelCloud: A Lightweight Labeling Tool for 3D Object Detection in Point Clouds, 2021. <http://doi.org/10.5281/zenodo.4591178>.
- [21] Sager, C.; Janiesch, C.; Zschech, P.: A survey of image labelling for computer vision applications. *Journal of Business Analytics*, 1–20, 2021. ISSN 2573-234X, 2573-2358. <http://doi.org/10.1080/2573234X.2021.1908861>.
- [22] Shreiner, D.; Board, O.A.R., eds.: *OpenGL programming guide: the official guide to learning OpenGL, version 2.1*. Addison-Wesley, Upper Saddle River, NJ, 6th ed ed., 2008. ISBN 978-0-321-48100-9. OCLC: ocn141853973.
- [23] Szeliski, R.: *Computer Vision: Algorithms and Applications*. Texts in computer science. Springer, London ; New York, 2011. ISBN 978-1-84882-934-3. <https://doi.org/10.1007/978-1-84882-935-0>. OCLC: ocn462920910.
- [24] Treiss, A.; Walk, J.; Kühn, N.: An Uncertainty-Based Human-in-the-Loop System for Industrial Tool Wear Analysis. In Y. Dong; G. Ifrim; D. Mladenić; C. Saunders; S. Van Hoecke, eds., *Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track*, vol. 12461, 85–100. Springer International Publishing, Cham, 2021. ISBN 978-3-030-67669-8 978-3-030-67670-4. [http://doi.org/10.1007/978-3-030-67670-4\\_6](http://doi.org/10.1007/978-3-030-67670-4_6). Series Title: Lecture Notes in Computer Science.
- [25] Veit, M.; Capobianco, A.: Go'Then'Tag: A 3-D point cloud annotation technique. In *2014 IEEE Symposium on 3D User Interfaces (3DUI)*, 193–194, 2014. <http://doi.org/10.1109/3DUI.2014.6798886>.
- [26] Wang, B.; Wu, V.; Wu, B.; Keutzer, K.: LATTE: Accelerating LiDAR Point Cloud Annotation via Sensor Fusion, One-Click Annotation, and Tracking. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 265–272, 2019. <http://doi.org/10.1109/ITSC.2019.8916980>.
- [27] Zheng, Y.; Zhang, D.; Xie, S.; Lu, J.; Zhou, J.: Rotation-Robust Intersection over Union for 3D Object Detection. In A. Vedaldi; H. Bischof; T. Brox; J.M. Frahm, eds., *Computer Vision – ECCV 2020*, vol. 12365, 464–480. Springer International Publishing, Cham, 2020. ISBN 978-3-030-58564-8 978-3-030-58565-5. [http://doi.org/10.1007/978-3-030-58565-5\\_28](http://doi.org/10.1007/978-3-030-58565-5_28). Series Title: Lecture Notes in Computer Science.
- [28] Zhou, D.; Fang, J.; Song, X.; Guan, C.; Yin, J.; Dai, Y.; Yang, R.: IoU Loss for 2D/3D Object Detection. arXiv:1908.03851 [cs], 2019. <http://arxiv.org/abs/1908.03851>. ArXiv: 1908.03851.
- [29] Zhou, Q.Y.; Park, J.; Koltun, V.: Open3D: A Modern Library for 3D Data Processing. arXiv:1801.09847 [cs], 2018. <http://arxiv.org/abs/1801.09847>. ArXiv: 1801.09847.
- [30] Zimmer, W.; Rangesh, A.; Trivedi, M.: 3D BAT: A Semi-Automatic, Web-based 3D Annotation Toolbox for Full-Surround, Multi-Modal Data Streams. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, 1816–1821, 2019. <http://doi.org/10.1109/IVS.2019.8814071>. ISSN: 2642-7214.
- [31] Zschech, P.; Sager, C.; Siebers, P.; Pertermann, M.: Mit Computer Vision zur automatisierten Qualitätssicherung in der industriellen Fertigung: Eine Fallstudie zur Klassifizierung von Fehlern in Solarzellen mittels Elektrolumineszenz-Bildern. *HMD Praxis der Wirtschaftsinformatik*, 2020. ISSN 1436-3011, 2198-2775. <http://doi.org/10.1365/s40702-020-00641-8>.