# Towards Object Classification using 3D Sensor Data

Sören Schwertfeger, Jann Poppinga, Max Pfingsthorn and Andreas Birk
School of Electrical Engineering and Computer Science
Jacobs University Bremen gGmbH
Bremen, Germany
Email: s.schwertfeger@jacobs-university.de

***Abstract*— This paper presents an approach to classify objects using 3D sensor data and an evolutionary algorithm. An important by-product of this classification is, that additionally certain properties and the pose in space of this object are determined. The Reproductive Perception Paradigm is used utilizing an evolutionary strategy. Two sub-approaches are discussed using different representations of the 3D data. The first one uses depth images while the second one uses point clouds stored in a special octree. The approaches will be demonstrated in experiments with simulated and real data.**

## I. Introduction

Object classification and recognition is a challenging and not yet fully solved problem in computer science and especially in robotics. The ability to reliably determine not only that but *which* objects are surrounding a robot permits it to deliberately reason and (inter-) act with its environment. This is especially true if not only the presence of certain objects can be detected but also their important properties like, for example, their pose in space. The proposed algorithm approaches this problem using the reproductive perception paradigm utilizing three-dimensional data and an evolutionary approach.

First the 3D sensor which was in mind when developing these algorithms, the SwissRanger, will be shortly introduced as well as the data-structure to store its readings, the FastOcTree (FOT). Next the reproductive approach using an evolutionary strategy and its fitness functions will be presented. The classification and simulated as well as real experiments conclude this paper.

## II. SwissRanger 3D data

The SwissRanger SR-3000 is a time-of-flight camera, i.e., a technology that is much less established than laser scanners or stereo cameras. An earlier version of this sensor is characterized in some detail in [1]. The technological principles on which this sensor is based are described in [2]. Roughly speaking, this type of sensor uses an array of cells similar to an imager of a camera to measure the phase-shift of emitted modulated infrared light. By this, a time-of-flight based distance measurement can be done simultaneously in each cell of the array. The sensor generates distance images as well as intensity images. The first correspond to the measured phase-shift, the second to the amplitude of the signal (see figure 1). One main advantage of this sensor is its large frame rate of up to 50 fps. The underlying technology of the SR-3000 is relatively young and far less established than laser range finders or stereo cameras. Though promising, there are still many drawbacks like the *wrap-around* error or the extremely small field of view, which is only $47^o \times 39^o$ due to the need of the LEDs to illuminate the scene. Other disadvantages of this sensor are the great sensitivity to ambient light conditions and reflections.

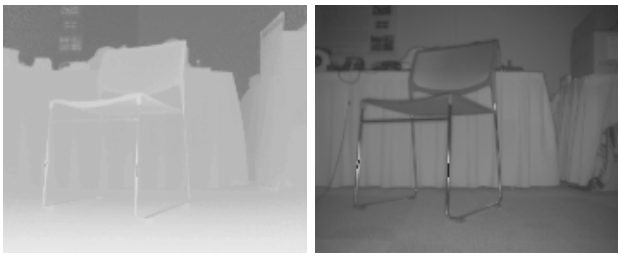

Fig. 1. Depth and Intensity Image of a chair taken with the SwissRanger

For the first approach depth images are used (see figure 1). For the algorithm the depth is encoded as a gray-scale value. Using straight-forward geometry 3D voxels can be computed from the depth image. All the valid pixel in the image form a point cloud which is being inserted into an octree for the second approach. Since the proposed algorithm only needs the binary information whether a cell is occupied or not a special fast octree implementation which also allows for very efficient nearest neighbor search has been developed.

## III. A Fast Octree Implementation

The well known octree data structure [3], [4] is a means of storing spatial occupancy information very memory-efficiently, especially more efficiently than in the default solution, a 3D grid, which also occasionally called regular mesh. More precisely, we are interested in a data-structure where for every cell $(x, y, z)$, a value $occ \in \{occupied, free\}$ has to be stored. The word "octree" is formed from "oct" (short for "octant") and "tree". As the name suggests, information is stored in a tree. An octree is able to *collapse* nodes. This happens when all sibling nodes are either "occupied" or "free". Then, this node will be represented in the parent by a single value, saving memory.

The FastOcTree (FOT) is an implementation of an octree optimized for speed [5]. The key design feature is the distinction between leafs of the octree and leafs of the FOT: The

**Algorithm III.1:** NEARESTTREENEIGHBOR($node$)

**procedure** NEARESTTREENEIGHBOR($node$)
**if** $node.hasChildren()$ **and not** at maximum depth
**then** $\begin{cases} node \leftarrow getNextChild(node, point) \\ \text{NEARESTTREENEIGHBOR}(node, point) \end{cases}$
**else** $upperBound \leftarrow 4 \cdot edgeLength(node)$
SEARCHCHILDEN($node, upperBound$)

**procedure** SEARCHCHILDEN($node, upperBound$)
**if not** $node.hasChildren()$
**then** $\begin{cases} upperBound \leftarrow \text{manhattan distance to filled} \\ \quad \text{cell closest to point} \\ nearestPoint \leftarrow \text{said cell} \end{cases}$
**else if** any point in this node falls in current upper bound
**then** $\begin{cases} children = \text{list of children, ordered by} \\ \quad \text{distance to point} \\ \textbf{for all } child \in children \\ \text{SEARCHCHILDREN}(child, upperBound) \end{cases}$

latter store the values for eight of the former in a bitmap, thus saving one level of nodes.

FastOcTreeNode is used for the nodes as well as for the leaves. It only has a single one-word member, a union. This union can either be used as a pointer or as a bitmap. As a pointer, it points to a struct containing all children as member variables. As a bitmap, it stores the occupancy of its children. Which role a node plays depends on whether it is a leaf or not, which can be determined by the value of the least significant bit of the union.

As an inner node the variable is a pointer. The class FastOcTreeNode is 4 byte memory aligned, thus ensuring that at least the two least significant bits of pointers to objects of this class are always 0. For FOT leafs, on the other hand, the union stores a bitmap. As an inner octree node always has eight children, only the most significant byte is used. The least significant bit is always set to 1.

The class FastOcTree manages the tree of FastOcTreeNodes. No coordinates and edge lengths are stored in the nodes, these have to be maintained by this class. Both iterative and recursive tree traversals are used, depending on the task at hand.

*1) Bresenham's algorithm:* In order to generate artificial 3D sensor data using 3D models, the Bresenham's line algorithm as well as a 2D Bresenham-like conic drawer are implemented for the FastOcTree. The Bresenham line and also the cone are very fast by using almost no multiplications, no divisions nor square-roots or trigonometric functions. Combinations of the line and the cone algorithm are used to occupy more complex primitives like boxes, cylinders or spheres.

*2) Nearest Neighbor:* Nearest Neighbor will be used to calculate the similarity between two FOTs. That means, given a particular cell in one data set, the closest cell with the same property must be determined in a second data set.

Due to the hierarchical structure of the octree, we can quickly find all filled cells and only compute the nearest neighbor distances for only those points. For this an algorithmic optimization is introduced, which extents the work of Hoel and Samet on nearest neighbor search on line segments in a quadtree [6]. In a dynamically growing octree, the presence of a node implies that within the boundaries of that node, there is at least one leaf node. Thus, we can immediately deduce an upper bound to the volume we have to search in the tree. Potentially, a lot of subtrees can be pruned from the search that lie completely outside this bound. While searching the nodes which do fall within this bound, we can progressively tighten said bound when we encounter leaf nodes in lower levels. This way we prune even more nodes from the search. Exploiting this property of octrees makes this algorithm very efficient. There are two phases for our recursive implementation of this algorithm as shown in III.1 : First, finding the octree node from which we want to start the search and finding the upper bound on the subsequent search. Second, we search this node's children and this node's parents children, and so forth, until we reach the root node. During the search, we progressively tighten the upper bound to correspond to the closest filled cell found so far, ensuring the pruning of subtrees that lie outside of the current bound.

In addition to these optimizations in the implementation, a special conceptual optimization is introduced, namely the so-called *LineBurstAccess*. This heuristic starts the traversal to access a specific node from the last visited one. This strategy is more efficient than the default to start from the root as geometrically close points are usually consecutively accessed.

## IV. REPRODUCTIVE PERCEPTION PARADIGM

Computer science approaches to perception are dominated by the view that perception is a process that takes large amounts of data from physical sensors like the pixel array of a camera and feed this data through various stages of processing that each lead to a reduction of the data. This holds especially for computer vision [7], [8], [9], [10], [11] but also with respect to more explicitly spatially oriented topics like map building [12]. The main idea of so-called reproductive perception is to do the opposite (figure 2) [13]. Perception is seen as a process, respectively a series of processes, where based on a small model large amounts of data are generated that match the incoming data from the sensors. The processes involved in perception so to say try to reproduce the data delivered by the sensors.

A (world-)model as a compact representation of the environment that is first generated and later on updated by perception is hence somewhat special within this paradigm. It is not a collection of passively descriptive data, but it can be thought of as a code in a kind of programming language that actually generates data.

So, a model is not constructed by stepwise processing of sensor data, but it itself generates large amounts of so-called pseudo sensor data, which is matched against the current sensor data. This generation of the pseudo-sensor data is denoted as rendering. By measuring the similarity between

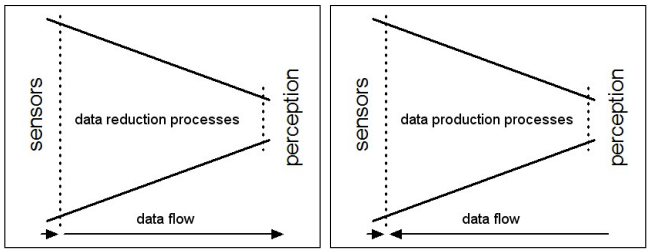


Fig. 2. Common computer science approaches to perception process sensor data in various stages that lead to a data reduction (left). The main idea of reproductive perception is in contrast that perception involves processes that generate data that matches the huge amounts of data delivered by the sensors (right).

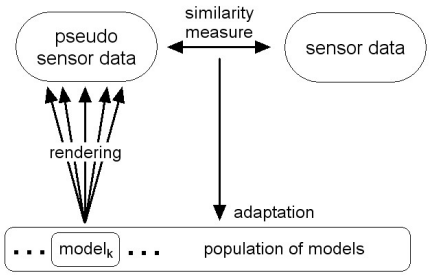


Fig. 3. The implementation of a reproductive perception presented here uses evolutionary learning. A population of evolving models is used to learn a representation that generates so-called pseudo sensor data, which should be as close as possible to the actual sensor data.

the actual sensor data and the pseudo sensor data, models can be generated and adapted.

This basic idea of reproductive perception is not tied to specific mechanisms. Nevertheless, a concrete implementation is presented here in form of an evolutionary learning scheme (figure 3). As mentioned, a model representing of the environment can be thought of as code in a special programming language. Models are hence programs that can be executed and that generate data. This data is then compared by a metric to the data from the sensors. This metric measures the similarity between the internally generated data and the data from the sensors. This measure of similarity can then be used as a fitness function in an evolutionary algorithm.

## V. Evolutionary Classification

The evolutionary approaches Evolutionary programming [14], Genetic Algorithms [15] and Evolution strategies [16] have been developed contemporary in the 1970's. Evolutionary Algorithms are search methods which draw inspiration from the natural search and selection processes leading to the survival of the fittest individuals. They use a probabilistic search mechanism which has a high probability of locating the global optimal solution although several local optimal solutions could exist as well [17].

### A. Experiments

In this section two fitness functions for 3D data are compared in two experiments. In these, artificial scenes have to be recognized. The goal scene to be recognized is given in a format suited for the used fitness function. For the evolutionary algorithm, each individual represents a scene. This scene usually consists of a number of object models. The population is evaluated by calculating a fitness value for every individual. This value is a measure for the similarity between the goal scene and the scene that the individual represents. Different similarity functions are used for the two approaches.

The algorithm for one evolution is summarized here:

```
Simple Evolutionary Algorithm () {
  randomly initialize population;
  evaluate population;
  while termination criterion not reached {
    select parents for next population;
    clone parents and perform mutations;
    evaluate population;
  }
}
```

The classification algorithms used here randomly initialize the population and then evaluate the population by applying a similarity function between the scene representation of the individuals and the goal scene. Parents for the next generation are chosen in the selection process and then cloned and mutated. The new population is evaluated and the loop is repeated until the termination criterion is reached, which in this case is the number of iterations. For the experiments 255 generations have been calculated. The roulette selection is used as selection method.

### B. Depth Image Fitness Function

In order to optimize for the goal scene, the algorithms use fitness functions which reward similarity of the image of an individual with the goal image. This is done by implementing a fitness function that only distinguishes between empty and occupied (by one of the primitives) voxels.

The depth image of the generated models is created by using OpenGL. The model is rendered into a scene and the view-port is set to the properties of the SwissRanger. For the depth images all the depth-distances between two corresponding pixel in both images are averaged. A fixed penalty for uncovered areas is used, where uncovered means error pixels or, for the generated models, background pixel. This value then directly represents the similarity where zero means perfect fit and higher values are worse.

### C. 3D Fitness Function

For computing the similarity between two octrees the nearest neighbor search is used. The similarity function is derived from the 2D image distance function $\psi$, which is based on accumulated minimal Manhattan distances between cells in the two data sets that share the same properties [18]. The 2D version defines the fitness between two 2D arrays $m_1$ and $m_2$ as:

$$\varphi(m_1, m_2) = \sum_{c \in C} d(m_1, m_2, c) + d(m_2, m_1, c)$$

$d(m_1, m_2, c) = \frac{\sum_{m_1[p_1]=c} min\{md(p_1,p_2)|m_2[p_2]=c\}}{\#c(m_1)}$
where:

- $C$ denotes the set of values assumed by $m_1$ or $m_2$,
- $m_1[p]$ denotes the value $c$ of array m1 at position $p = (x, y)$,
- $md(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$ is the Manhattan distance between points $p_1$ and $p_2$,
- $\#c(m_1) = \#\{p_1|m_1[p_1] = c\}$ is the number of cells in $m_1$ with value $c$.

In the 3D version the Manhattan-distance is now defined as: $md(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|$.
The most important difference to the 2D approach is, that the distance map is not used. Generating a distance map would mean, that the whole octree would have to be expanded, which is extremely expensive both computationally and with respect to memory size. Instead the Manhattan-distance is calculated for all voxels by searching in the FastOcTree. The special structure of the octree is, as already mentioned, very helpful for finding upper boundaries for the next occupied cell, which allows, together with other optimizations, a fast search for the nearest occupied voxel. Once the voxel is found the Manhattan distance is easily calculated using the above formula.

### D. The Classification

The experiments all define different classes by placing three primitive objects in one scene on position one, two and three. There are thus 216 different classes (three primitives of six different shapes). The primitive objects have attributes which vary from individual to individual, for example the size, the orientation and the position. Two scenes are regarded as being in the same class if their first, second and third primitives are from the same class, although their attributes can be quite different. Those primitive classes are box, ball, cross, cylinder, T-shape and L-shape. The scene does not only differ in the objects themselves but also in the position, the size and the rotation of each primitive - within certain bounds (see figure 4). Due to this freedom the primitives have a certain chance to overlap.

The classification tests every possible class with the evolutionary algorithm. For each of these classes, one evolution is started in which the population is initialized with individuals from this class. The evolution then mutates the attributes of these scenes such that an optimal fit between the representation of the goal scene and the scenes in the population is reached. The fitness value of the best individual is then taken as the similarity value between this class and the goal-class. The similarity is calculated for each class. The class with the highest similarity is then regarded as the class in the goal scene (provided that it reached a sufficient similarity value).

For each experiment 400 evolutions were run. In 200 tests the populations were initialized with scenes from the same class as the goal-class, while in the other 200 runs at least one of the primitives was different. This great number allows for good statistical proposals.

In order to actually classify we specify a similarity value threshold which evolutions of a class have to reach to be classified (see figures 6 and 8). This threshold value $b$ was computed using the experiments and is defined to be somewhat in the middle between the average correct and the average false example:
$b = (v_{AverageFalse} - v_{AverageCorrect}) * 0.3 + v_{AverageCorrect}$
Now all results are compared to this threshold and for every single evolution the class is defined.

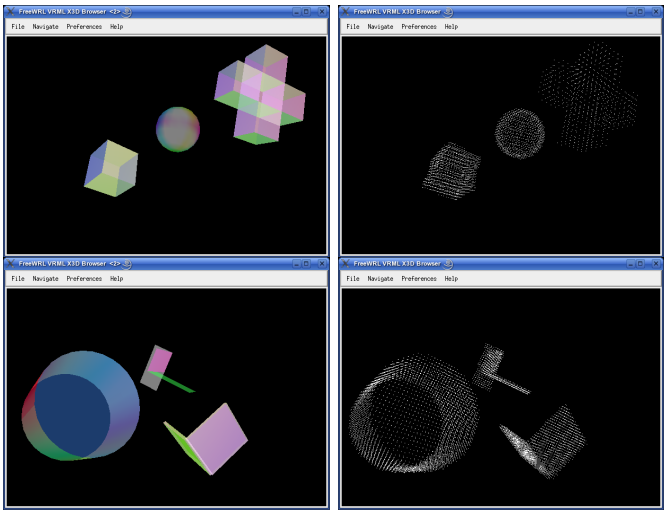

Fig. 4. On the left side the primitives are visualized. On the right side the respectively occupied cells of the octree are shown as a point-cloud.

## VI. Simulation Experiments

For these experiments a special representation was developed which is capable of multiple output formats. The shapes can be rendered with OpenGL and exported as X3D and in text format. The most important feature of this representation is, that it can render the surface of those models into a FOT in order to use the presented 3D similarity as fitness function.

As mentioned there are 216 different classes with which the individuals can correspond. The problem to be solved is to identify the class of such a (randomly generated) individual. That means that the input available for the program is just (simulated) goal sensor-data of this goal individual.

During evolution the primitives of the individuals inside the population will adjust their size, position and their rotation to the goal scene. But only individuals which are in the same class as the goal can optimize their values such that a perfect fitting value can be achieved. Those individuals are then exact models of the goal and provide, in addition to the class of the goal, also information about the position, size and rotation of its primitives. The Evolutionary Strategy that is being used is a basic version without any improvements like adoption of the mutation strength.

The initial population always consist of individuals which are all in the same class. Two different scenarios were performed. In the first test run the initial class was the same as the goal class, in the second test one of the primitives always differed from the goal individual. In figure 5 the development of the fitness values of an example evolution is displayed

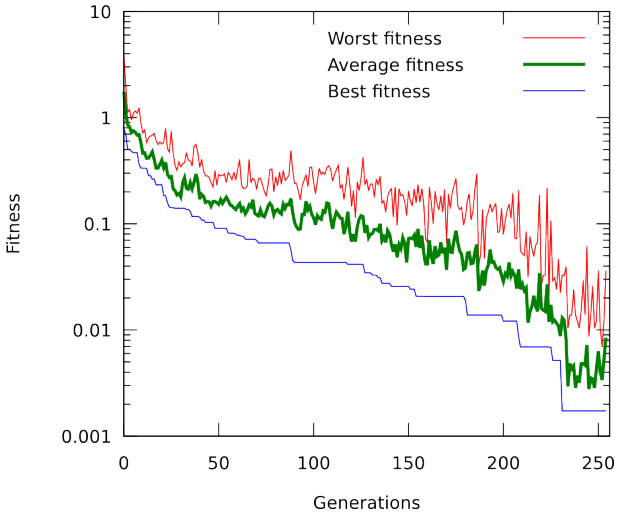


Fig. 5. Development of the fitness during an example evolution

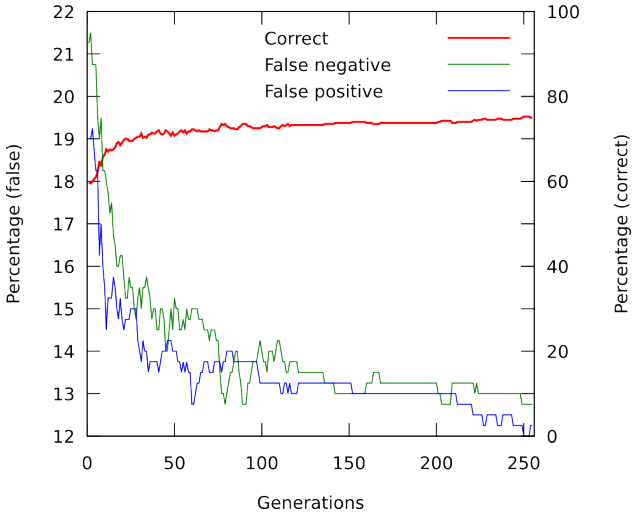


Fig. 7. Depth image approach: Classification results

(please note the logarithmic scale on all diagrams). Best fitness shows the development of the fitness of the best individual. The graph is convergent because the plus selection strategy was implemented which keeps the selected parents in the population. Average fitness is the average fitness value of all individuals in the population while Worst fitness is the value of the worst fitness in the current population.

The population size is 30. Six parents plus the best parent are selected for breeding using the roulette selection method.

### A. Results of the depth image approach

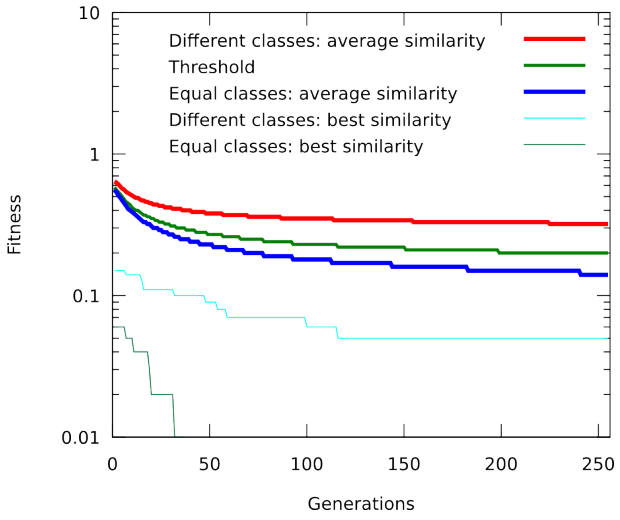


Fig. 6. Depth image approach: Comparison with same and different shapes

The results for the evolution of the depth image approach can be seen in figure 6. It shows the average and best (of the 200 runs) similarity ($\hat{=}$ fitness) between the goal individual and the generated individuals - once for the case where the classes are equal and once where they differ. The threshold that is used for classification is also shown. The results of the classification are visualized in figure 7: over 70% are correct after 100 generations. After 255 generations 75% are reached with about 12% false positives and about 12% false negatives. As can be seen in the figures the classification quality reaches considerable levels fast and then improves only slowly.

### B. Results of the Octree Approach

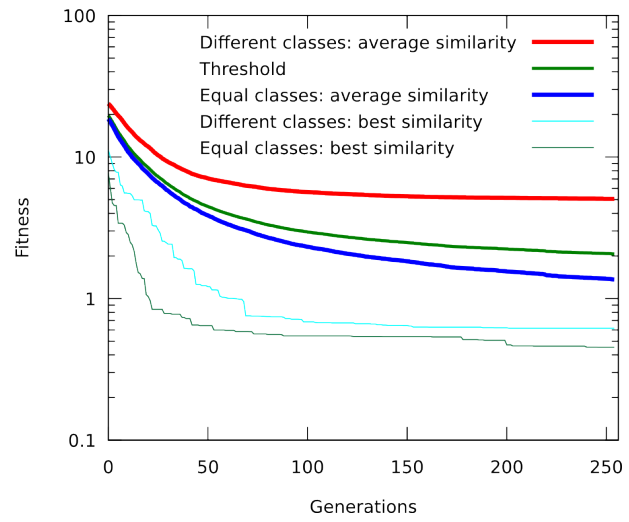


Fig. 8. Octree Approach: Comparison with same and different shapes

The results for the Octree approach (figures 8 and 9) are slightly better then those from the depth image algorithm: After 100 generations the classification is correct in about 75% of the cases while this number reaches over 86% after 255 generations.

## VII. Conclusion and Outlook

The experiments show that both approaches achieve quite good results. The 2D depth image approach computes fast but

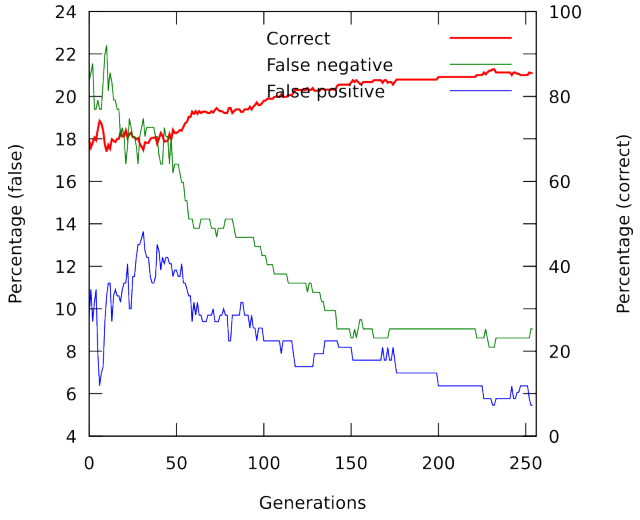


Fig. 9. Octree Approach: Classification results

relies on the fact that the 3D data is present as a projection of 2D data. It can thus not be used for general point clouds. Being able to work on general point cloud data is one main advantage of the 3D FOT approach. Thus it can use point clouds generated out of depth images taken from different positions or from actuated LRF. This way occlusions are even less a problem as they are already in the reproductive approach in general.

### A. First real world results

In future real world data will be used to test and compare both algorithms - a first experiment can be seen in figure 10. Possible objects to be classified are, for example, stairs, chairs, tables, cupboards or humans. These classification algorithms go nicely hand in hand with 3D mapping algorithms. They can replace simple planes generated for the classified objects in the 3D map with their models, thus making the map more accurate and readable.

The data of the stair for figure 10 was gathered with the SwissRanger. A point cloud which has already undergone some preprocessing to reduce the noise can be seen in which a model of a stair has been fit into by the 3D evolutionary classification algorithm. The algorithm not only detected the presence of a stair but also its spatial pose and properties like number of steps, the step width and height and depth as well as the overall inclination of the stair.

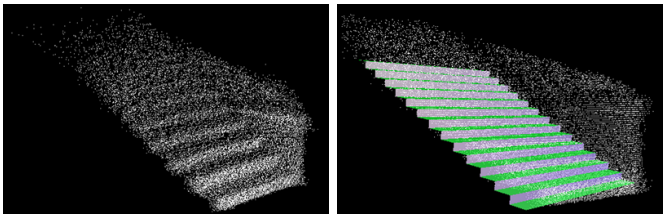

Fig. 10. The point cloud of a stair gathered with the SwissRanger and a stair model created with the 3D reproductive perception approach.

## ACKNOWLEDGMENT

The authors gratefully acknowledge the financial support of the *Deutsche Forschungsgemeinschaft* (DFG).

Please note the name-change of our institution. The Swiss Jacobs Foundation invests 200 Million Euro in **International University Bremen (IUB)** over a five-year period starting from 2007. To date this is the largest donation ever given in Europe by a private foundation to a science institution. In appreciation of the benefactors and to further promote the university's unique profile in higher education and research, the boards of IUB have decided to change the university's name to **Jacobs University Bremen**. Hence the two different names and abbreviations for the same institution may be found in this article, especially in the references to previously published material.

## REFERENCES

[1] J. Weingarten, G. Gruener, and R. Siegwart, "A state-of-the-art 3d sensor for robot navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3. IEEE Press, 2004, pp. 2155–2160 vol.3.

[2] R. Lange and P. Seitz, "Solid-state time-of-flight range camera," *Quantum Electronics, IEEE Journal of*, vol. 37, no. 3, pp. 390–397, 2001.

[3] C. L. Jackins and S. L. Tanimoto, "Oct-trees and their use in representing three-dimensional objects," *Computer Graphics Image Process*, vol. 14, no. 3, pp. 249–270, 1980.

[4] D. Meagher, "Geometric modelling using octree encoding," *Computer Graphics Image Process*, vol. 19, no. 2, pp. 129–147, 1982.

[5] J. Poppinga, M. Pfingsthorn, S. Schwertfeger, K. Pathak, and A. Birk, "Optimized octtree datastructure and access methods for 3d mapping," in *International Workshop on Safety, Security, and Rescue Robotics (SSRR)*. IEEE Press, 2007.

[6] E. G. Hoel and H. Samet, "Efficient processing of spatial queries in line segment databases," in *Advances in Spatial Databases, Second International Symposium, SSD'91, Zürich, Switzerland, August 28-30, 1991, Proceedings*, ser. Lecture Notes in Computer Science, O. Günther and H.-J. Schek, Eds., vol. 525. Springer, 1991, pp. 237–256.

[7] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 399–458, 2003.

[8] M.-H. Yang, D. Kriegman, and N. Ahuja, "Detecting faces in images: a survey," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 1, pp. 34–58, 2002.

[9] L. G. Brown, "A survey of image registration techniques," *ACM Computing surveys*, vol. 24, no. 4, pp. 325–376, 1992.

[10] D. Vernon, *Machine Vision*. Englewood Cliffs: Prentice Hall, 1991.

[11] B. K. P. Horn, *Robot Vision*, ser. MIT electrical engineering and computer science series. MIT Press, Cambridge, 1986.

[12] S. Thrun, "Robotic mapping: A survey," in *Exploring Artificial Intelligence in the New Millenium*, G. Lakemeyer and B. Nebel, Eds. Morgan Kaufmann, 2002.

[13] A. Birk, "Spatial knowledge processing within the reproductive perception paradigm," in *Control Mechanisms for Spatial Knowledge Processing in Cognitive / Intelligent Systems*, ser. Spring Symposium. Stanford: AAAI, 2007.

[14] L. Fogel, A. Owens, and M. Walsh, *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

[15] J. H. Holland, *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

[16] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog, Stuttgart, 1973.

[17] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *Computer*, vol. 27, no. 6, pp. 17–26, 1994.

[18] A. Birk, "Learning geometric concepts with an evolutionary algorithm," in *Proc. of The Fifth Annual Conference on Evolutionary Programming*. The MIT Press, Cambridge, 1996.

Schwertfeger, S., J. Poppinga, and A. Birk, "Towards Object Classification using 3D Sensor Data", ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS): IEEE, 2008.

http://dx.doi.org/10.1109/LAB-RS.2008.28