DUPLOG: Probabilistic Logical Interpretation of Duplo Assemblies from 3D Vision

Sil van de Leemput [a] Martijn van Otterlo [a]

[a] *Artificial Intelligence – Radboud University Nijmegen, The Netherlands*

Abstract

Composite real world objects can be logically explained by their components and their relations. Knowing and expressing the logical relations between the components can facilitate 3D object recognition employing semantic information. We present DUPLOG, a novel method for 3D model acquisition of Duplo block assemblies from video. It acquires percepts trough a Kinect camera and translates these inputs into a structured representation through octrees and logical constraints. The high-level, declarative (logical) constraints are represented in a probabilistic logic and provide a new principled way for 3D object recognition. We show that our system is efficient and effective in the interpretation of arbitrary Duplo buildings.

1 Introduction

Interpreting visual scenes is a hard task and the field of computer vision [12] has developed many techniques for segmentation, classification, recognition and retrieval of images, objects and scenes. Currently various 3D object recognition techniques exist as well, among which: affine invariant image, object silhouettes, object attributes and stereo vision. Even with all these techniques 3D vision remains a tough problem.

Humans are very good at interpreting and understanding the 3D world around them through visual input. They do this typically by employing *knowledge* about the world to make sense of what they *see*. A car is more likely to be found on a road than in the water, houses have doors and windows, if there is fire then there is smoke, etcetera. Utilizing declarative, high-level knowledge is highly desirable in computer vision, although the inherently noisy and uncertain nature of vision data has been an obstacle to actually applying such knowledge in vision contexts. Yet, a large set of vision problems is about structured objects.

Let us take a structured domain, for example imagine playing with Lego, the colored building blocks everyone will recognize immediately. By clamping blocks on other blocks a very large amount of assemblies can be created. While building and interpreting, humans use a lot of knowledge on how the blocks fit onto each other, which structures are possible, which hidden structures are more likely given the number (and types) of available blocks left, and so on. An automated system doing the same thing would need to acquire images from an assembly of Lego blocks in order to interpret it, but noise in measuring, uncertainty in color detection, distortion of angles and vanishing points, and many other challenges arise. However, high-level knowledge on the *types* of objects, as well as their possible (spatial) *relations* may help in interpreting such noisy images. For example, if some part of a building is occluded, logical reasoning about possible structures may give rise to hypotheses of what could be missing. Also, a shadow which changes the color of some part of a block may be explained away by logical reasoning on the neighboring color features.

In this paper we introduce a novel technique to do exactly the sketched situation: we present an automated system that can interpret [1] actual Lego assemblies from camera input. For practical reasons we use the slightly larger Duplo blocks, but this represents no conceptual difference. Very few related works exist

[1] Effectively we have created a 3D version of Lego's own (2D) *Life of George* game.

in this Duplo/Lego domain. However our system is novel by taking a principled approach incorporating high-level interpretation steps using a probabilistic logical reasoning system. The use of logic allows for simple and effective specification and extension of a knowledge base of *facts*, *rules* and *spatial relational theories* about the objects that have to be recognized. In addition, our system is a *general* technique for 3D object recognition and could be applied to many other domains.

Our system, named DUPLOG, employs a commonly used Kinect [2] camera to acquire visual input. This camera was introduced in 2010 as part of the XBOX gaming system, but has since been adopted by the computer vision (and robotics) community as an affordable and highly effective 3D camera system. Existing software can extract so-called *point clouds* from a visual scene which represent distances from the camera to any surface in the current visual view. By adding (appearance-based) color information about surfaces, and by combining multiple views of the same object, full point cloud models of an object can be obtained. In addition, we employ existing software to obtain *octree* representations to abstract such point clouds into a tree-based representation of a 3D grid containing the points in the point cloud. From there, logical reasoning is used to interpret the current assembly of Duplo blocks. To deal with the inherent noisy and uncertain outcome of the previous steps, we employ a probabilistic version of Prolog.

Our main **contributions** concerning DUPLOG are the following: i) a principled way of coupling computer vision with probabilistic logic, mediated by octree representations, ii) a novel technique for structured, 3D interpretation of Duplo assemblies, and iii) a working application capable of interpreting full Duplo assemblies in under ten seconds on standard hardware.

The **outline** of this paper is as follows. We first discuss related work in Section 2, after which we introduce our system in Section 3, consisting of a computer vision part (see Section 4) and a probabilistic logical part (see Section 5). Experimental results are shown in Section 6 after which Section 7 concludes and points to future research directions.

2 Related Work

Our work is situated at the crossing between computer vision, structured (logical) representations of objects, and the Duplo/Lego domain. Our system is built upon the idea that many visual scenes are best described using high-level representational devices such as graphs, and even more generally using logical languages.

Despite much research, intelligent, practical and real-time object detection and recognition methods are still not present for the general case [11]. General computer vision [12] has matured enormously, but structured (logical) representations (see [10] for an overview) are less represented in the vast literature.

Older work in *structured* (or: *syntactic*) *pattern recognition* [2] already acknowledged the usefulness of structured representations (such as graphs) for visual data, but lacked the power and flexibility of modern *probabilistic* reasoning systems for such structured data (see [3] for good starting points to the literature).

Increasingly many examples of using high-level, structured representations in computer vision appear in the literature in recent years: high-level scene perception using description logics [8] or using a hierarchical approach [1]. A recent trend in the neighboring field of *robotics* is to employ such representations for *affordances* [7], i.e. representing objects in terms of *what one can do with them* (for example, a chair *affords* one to sit on it). These systems are based on similar visual input, and utilize structured representations of recognized objects for subsequent tasks.

Concerning the Lego domain, some papers aim to *generate* structures, for example using a (population-based) search method. Peysakhov et al. [9] describe a representation of Lego block assemblies which we adopt for the current paper. For the *interpretation* of assemblies from visual input, two very recent approaches exist. Both KINECTFUSION by Izadi et al. and the work of Miller et al. [6, 4] target this problem. Their systems are able to capture percepts in real-time and build a model of scene objects in real-time. The work of Miller et al. is especially relevant for this paper, since it shows an efficient way to acquire a model of Duplo block assemblies by introducing some constraints. They also describe improvements in the combination of multiple percepts in an integrated model at real-time. But both methods do not utilize our unique combination of probabilistic logic and octrees to acquire the models.

[2] More on the Kinect (for Xbox 360): `http://www.xbox.com/Kinect`

3 Problem Specification and Approach

On a very high level, we want our system to acquire the correct representation of real world objects over time, given logical constraints on how (sub)parts of the objects may relate to each other, for example in a *spatial* sense. Our restricted domain of objects here consists of Duplo [3] assemblies using an arbitrary maximum number of Duplo blocks. Hence, our **problem statement** can be formulated as follows.

> **Given** an input consisting of raw image data containing footage of an arbitrary Duplo block assembly A, acquired from a Kinect camera, **construct** a logical representation $R(A)$ in a representation like the one used by Peysakhov et al. [9], **making use of** a logical *background theory* about (spatial) relations and constraints about Duplo blocks.

To tackle this problem, we need to pay attention to several important aspects which we describe here briefly. After that we present a general flow diagram of the various steps in the interpretation process and explain the two main sub-steps in more detail in the next sections.

Input and Output We restrict our input to be all assemblies that can be made from a fixed (and known) set of (currently nine) Duplo blocks. This allows for (at least) a couple of hundreds (for humans) visually different assemblies one can construct. We assume that assemblies are placed flat on the grid surface, all studs facing up. All the surfaces of the Duplo blocks have to be perpendicular or parallel to each other. As output we take a structured representation which specifies where each block is in space. It should be able to correctly reconstruct the input Duplo assembly from that representation. We leave open that, since the percepts can only capture the visible surfaces of a configuration, the *inside* of an assembly may lead several hypotheses about what is there. It is only reasonable to expect that the output captures the *outside*.

Logical constraints Constraints on Duplo blocks should be interpreted as a set of *rules*. These rules can represent either properties of blocks, or specify how blocks are (spatially) related. The first type of rules include: *blocks should always have at least four flat surfaces* and *blocks can only be red, green, blue or yellow*, etc. Other rules include: *when a red surface is detected it could be composed of multiple red blocks*.

From Vision to Logic Since we start from raw image data, and finally (want to) arrive at a logical representation, at some point we need to bridge the gap between *low-level data* and *high-level interpretations*. In our system we mark the boundary between these sub-systems by representing a summary representation of the underlying low-level data using *logical facts*. This representation should be chosen wisely, since it needs to be both the output of the low-level vision phase and the input to our logical interpretation system. Here we compute these logical facts (i.e. features) directly from the *point clouds* and *octrees*.

Point clouds and octrees The depth and color images that the Kinect provides can be represented as a set of points (x, y, z, r, g, b). Here the x, y, z values indicate a position in Euclidean space and the r, g, b values stand for the color intensities of respectively red, green and blue. This representation is commonly known as a point cloud, where each point is called a *voxel*.

Since points are a too noisy, and too detailed representation for regular structures such as Duplo blocks, we use an abstraction known as *octree* [5]. These can segment a detailed point cloud into a more compact representation, downsampled to a specified resolution. Octrees are, as the name suggests, tree representations that have for each non-leaf node eight sub-nodes. Each node maps to a cube-like space segment and each sub-node maps to one of eight equally-sized sub-segments of that cube. When a node is only segmented if it actually contains points, a *sparse* octree can be obtained. Advantages of such a sparse representation is the facilitation of point localization, the provision of additional information about the raw outline of the points, and faster operations on the trees.

Each leaf node in the (sparse) octree represents a part of space occupied with (visual) points found in the point cloud. It is here where the step towards a logical representation is made. Each such leaf node, called a *bin* is represented as a logical fact, along with information about its location in 3D space as well as (average) color information of the points it contains. Informally we can say that each bin represents a small piece of physical matter detected by the camera. Using logical reasoning with constraints, we can then *group* several of these pieces (i.e. bins) to form Duplo blocks.

[3] More on Lego (and Duplo): `http://www.lego.com/`

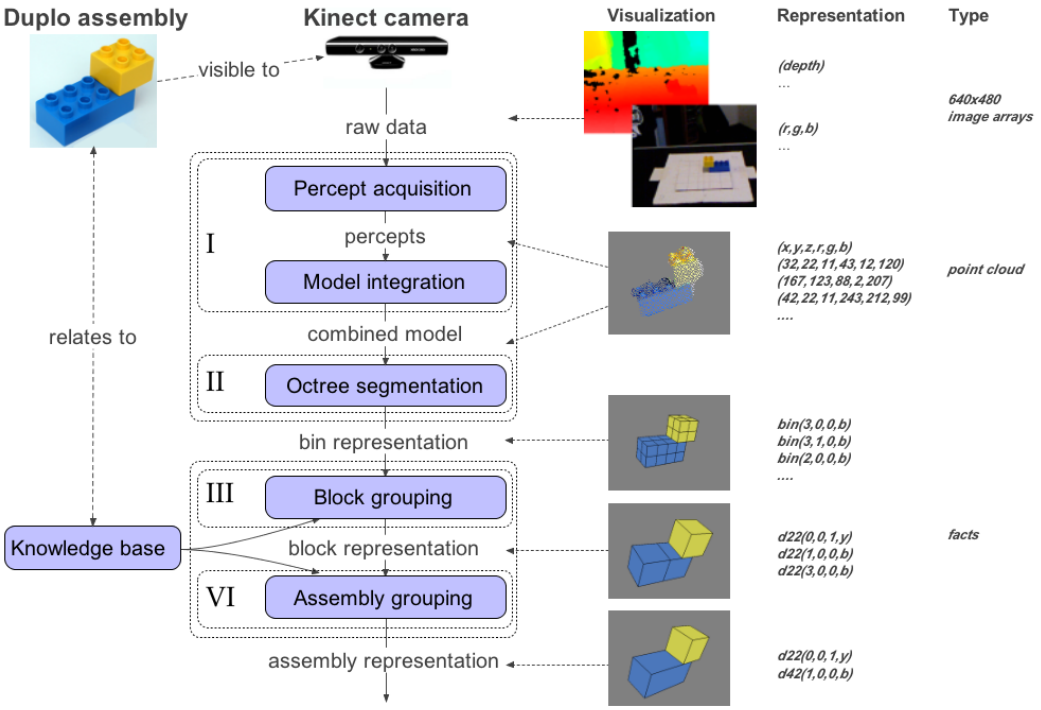

Figure 1: Global overview of the components and associated representations of the method.

Overview Figure 1 shows the complete interpretation process in DUPLOG. First raw "real world" percepts from a Duplo block assembly come from the Kinect camera (at the top) that outputs both depth and color images. The representation will be transformed into point clouds, using filters and corrections. Since one point cloud percept only gives voxels from one side of a Duplo block assembly it is necessary to then combine multiple percepts into a single point cloud (completing step I). After multiple percepts are combined octree segmentation (step II) can be used to prepare for the logical part of the method, by translating a point cloud to an octree and representing the bins of the octree as logical facts. Steps I and II represent the vision part of the system and are discussed in Section 4.

For the next step the bin facts are translated into 2×2 Duplo block predicates (step III). Finally the 2×2 Duplo block predicates are translated into the "correct" output assembly representation (step IV). The last two steps use a knowledge base that contains Duplo blocks related constraints to calculate the "correct" output assembly representation. Both steps are discussed in Section 5.

4 Feature Extraction from Visual Input

Our system only considers Duplo assemblies on a grid on an uncluttered tabletop surface (see Figure 2(right)). This greatly simplifies the 3D model acquisition problem using point cloud models [4]. The physical grid relates to a 3D bin grid G in model coordinates. This grid G can be described as a 3D volume segmented into $N_1 \times N_2 \times N_3$ sub-volumes called bins, with $N_1, N_2, N_3 \in \mathbb{N}$. One bin in the grid has dimensions d_x, d_y, d_z that corresponds to one out of eight parts of a 2×2 Duplo block (excluding the studs), see Figure 2(left). G can be interpreted as a container for all points of the directly visible surfaces. The depth and width for a bin is the same for the X and Z axes (implying: $d_x = d_z$). Our system treats the center of the grid surface as the

[4] We use the C++ version of the Point Cloud Library for the point clouds and octrees: `http://pointclouds.org/`. For the Kinect camera we use the OpenKinect library, see `http://openkinect.org/wiki/Main_Page`.

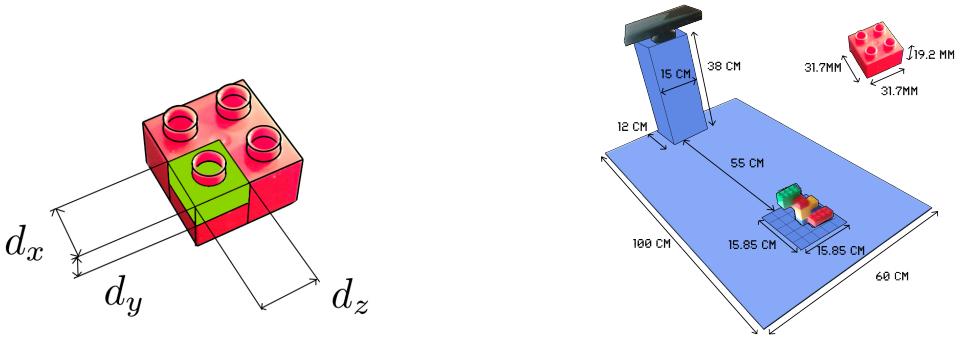

Figure 2: (left) Size of a bin relative to a 2×2 Duplo block, (right) the physical setup.

origin. The Y-axis is assumed to be perpendicular to the grid surface.

Step I: Percept acquisition and model integration
A point cloud model containing all visible *surfaces* for one Duplo block assembly can be obtained by taking point clouds captured from different viewing angles; in our case four. Multiple raw percepts (in physical coordinates) generate a combined model P^{cmb} (in model coordinates), using

$$P^{cmb} = \bigcup_{i=0..3} C_i R T P_i$$

where P_i are the raw percepts point clouds existing of N voxels in physical coordinates. Here T is a transformation matrix and R is a rotation matrix that together align P_i to the 3D bin grid G. C_i is a discrete rotation matrix that rotates all points around the Y-axis with a multiple of $i \times 90°$. T, R and C_i are constant matrices and can be precomputed. By simply adding all the points from the aligned point clouds we obtain P^{cmb}. But, since the aligned point clouds were taken from different viewpoints we first rotate each by a multiple i of $90°$ corresponding to the amount of times the model was physically rotated around the Y-axis. This gives us the discrete correction matrices C_i with a different rotation around the Y-axis for each i.
Preprocessing The input from the sensor are two 640 x 480 images, one with color information and the other with depth information. This input is transformed to a Euclidean point cloud P_i so that the perspective is orthogonally corrected and the unit size is expressed in millimeters. The calibration measurements to do this may be provided for the sensor or can be determined experimentally. The calibration corrections also map the two images correctly on each other so that for each voxel in the point cloud P_i 3D points there is an associated color value. A plane is positioned behind the grid surface to segment the interesting voxels (the Duplo assembly) from the background.
Grid alignment The point clouds P_i contains voxels indicating *surfaces* of Duplo blocks. In order to segment the point clouds P_i, we need to align the surfaces of the point cloud P_i to the 3D bin grid G. In other words: we need to find a translation matrix T and a rotation matrix R such that when applied on point cloud P_i it is correctly aligned with G.

The rotation matrix R has to make sure that all the surfaces for the blocks are either perpendicular or parallel to the bins, taking into account earlier described constraints. The translation operation T should make sure that the surfaces of the point cloud are relatively the same distance from the model origin as the surfaces of the Duplo blocks to the origin in the real world. The trick with this last step is not to exactly align the surfaces on the outlines of the cubes of the octree, but to move all voxels a little further down the Z-axis (no more than half a bin). In this way the occupied bins represent found *surfaces of Duplo blocks*. Our octree structure uses only cubes, i.e. $d_x = d_y = d_z$. We multiply the Y-coordinate with a factor $\frac{d_x}{d_y}$ to satisfy this condition. We also discard all voxels with an Y-coordinate lower than 0 (below the origin).

Step II: Octree segmentation
Next we segment the combined point cloud model P^{cmb} using an octree and return the relevant octree bins and their associated color. Essentially this step segments the combined model with the 3D bin grid G, with

as smallest unit $d_x \times d_y \times d_z$. This renders each 2×2 Duplo block being segmented into eight bins. Eight instead of four bins resulted in a lower overall error rate. A single bin within the octree contains all voxels from P^{cmb} within its bounds. Many bins will contain some voxels but are not interesting for the Duplo assembly structure. Such irrelevant bins can be discarded using a (empirically validated) threshold $t_{bins} = 60$: the minimal amount of points a bin needs to contain to be considered. We estimated the qualitative colors (red, green, blue, yellow) by taking the color values of the median voxel in a bin and calculating the relative distance between the color values for that bin and the color values for each qualitative color. If this distance exceed the threshold $t_{color} = 50000$ it was ignored. Since lighting conditions may change easily per environment, our system supports calibration using measurements of blocks of each color.

5 High-level Interpretation using Probabilistic Logic

Steps I and II have transformed the raw image input into a set of *bins*, each representing a part of the point cloud of the underlying data. We first represent these bins as *logical facts*: $\texttt{bin}(3,0,0,\texttt{b}), \texttt{bin}(3,1,0,\texttt{b}), \ldots$, (see also Figure 1) where the first three arguments determine the position of the bin in 3D space, and the last contains one of the colors of the Duplo domain. Our next two steps assemble *groups* of bins into Duplo blocks. One option is to use *abduction*, trying to *explain* the bins using *hypotheses* consisting of assemblies of Duplo blocks. However, our target Prolog platform, the probabilistic Prolog variant PROBLOG [5], does not support our modeling of this setting directly, hence we employ a *deductive setting* instead. We do not use PROBLOG to calculate a full probability distribution, but we take the first grouping solution with the highest probability using the *problog_max* predicate, i.e. we apply a greedy search. The first step (III) will generate a Duplo assembly consisting of only 2×2 blocks, i.e. of the smallest granularity. The second step (IV) tries to assemble 2×2 into larger 4×2 blocks if available in the system and likely present.

Step III: Block grouping

The bins are grouped into blocks first: one *block* consists of eight bins within a $2 \times 2 \times 2$ bin cube. The block grouping is efficiently done in a sequential fashion from the two highest layers of bins to the two lowest layers of bins and per color, see Algorithm 1. The resulting block-set R' for $mostLikelyGrouping(D', C_1, P_1)$ is determined by a Problog program (C_1, P_1) by (greedily) taking the most likely block facts for bin-set D':

$$D', C_1, P_1 \vdash_{\text{problog\_max}} R'$$

The constraints C_1 and probabilities P_1 enforce that each bin in D' can either be a part of a block with probability $p_{block} = 0.95$ and missing bins have a probability $p_{false\_negative} = 0.2$. Alternatively a bin can be noise with probability $p_{false\_positive} = 0.02$. Finally a list with the highest probable block grouping is returned as a block representation R'. The probabilities for this step were estimated based on initial tests.

input : A list B of bins $(x, y, z, color)$
output: A list R of 2×2 blocks
begin
  $R \leftarrow \emptyset$
  while $B \neq \emptyset$ **do**
    $D \leftarrow$ take $S \subseteq B$ where S contains exactly all bins of the highest *block* layer for B and no other bins
    for *each* $color \in r, g, b, y$ **do**
      $D' \leftarrow$ take $S \subseteq D$ so that S contains exactly all bins with color $color$ and no other bins
      if $D' \neq \emptyset$ **then**
        $R \leftarrow R \cup mostLikelyGrouping(D', C_1, P_1)$
    $B \leftarrow B \setminus D$
  return R

Algorithm 1: Block grouping.

Step VI: Assembly grouping

The last step takes the step III's results and computes the most likely assembly interpretation S in the form of a list of Duplo blocks given Duplo related constraints C_2. The list of blocks R from step III and experimentally estimated probabilities $P_2 = \{p_{d22} = 0.60, p_{d24} = 0.15, p_{d42} = 0.15, p_{noise} = 0.01\}$ for grouping occurrence determine an optimal solution S (having the highest probability) as:

[5] Information on Problog: `http://dtai.cs.kuleuven.be/problog/`

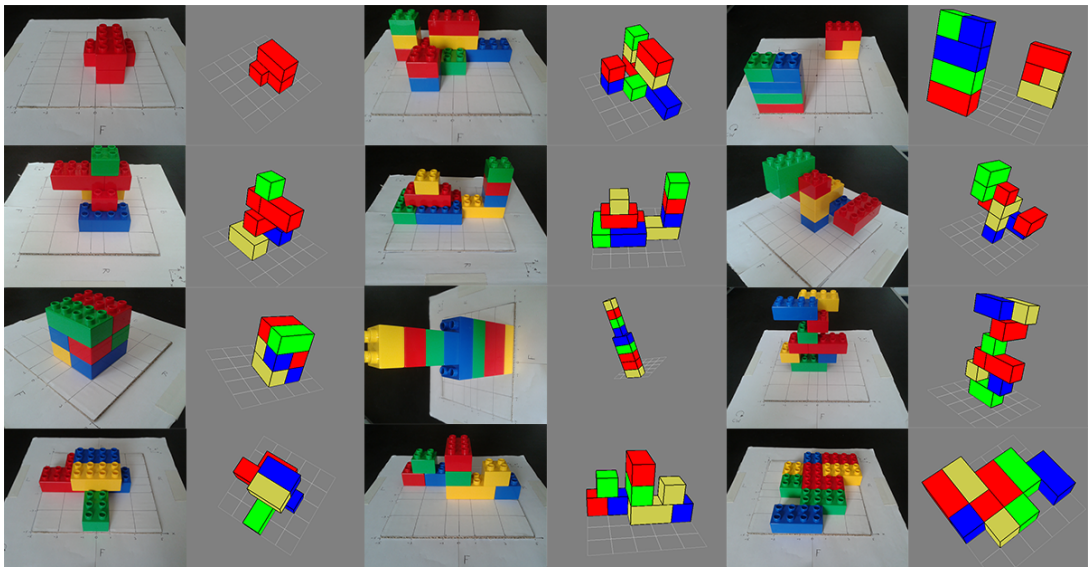

Figure 3: Series (3×4) of examples. Each shows an image of the original assembly (left) and the corresponding 3D interpretation computed by DUPLOG.

$$R, C_2, P_2 \vdash_{\text{problog\_max}} S$$

The constraints C_2 that where used for this step are: **i**) blocks come from a fixed/known set, **ii**) Duplo blocks can not overlap each other, **iii**) all blocks with $y = 0$ are on the ground, **iv**) blocks need to be supported by other blocks if they are not on the ground (meaning that it can not float in the air on itself), **v**) two bordering blocks can be grouped as one 2×4 or one 4×2 Duplo block with respectively probabilities: p_{d24} and p_{d42}, and **vi**) a block can be noise or be a 2×2 Duplo block with respectively probabilities p_{noise} and p_{d22}.

6 Experiments and Results

A simple use-case of DUPLOG works as follows. The system is first calibrated for correct lighting conditions and is set to the origin. The user constructs a Duplo assembly and places it on the grid. In the capturing phase, the user rotates the assembly three times to let the camera capture the four viewpoint point clouds. Finally the system calculates an assembly representation from the captured input and shows it to the user.

Examples The program was tested against a database of 51 combined (i.e. after steps I and II) Duplo models. For 78% of them the accuracy is 100% (see Figure 3 for some examples). 6% of the models were not acquired correctly, hence incorrectly identified. The other 16% were only partially correct; this mainly occurs because of occlusion. This makes that the octree segmentation only represents a few of the outer bins for the occluded blocks or even no bins at all. The block grouping does not take the possibility of occluded blocks into account and therefor results in producing "floating" blocks. The last logical step then will interpret the "floating" blocks as noise, while they actually are supported by an obscured block. This is easily solved by extending the logical interpretation program accordingly, but we leave that for future work.

Statistics Our program runs on Mac OS X 10.6.8, Intel Core 2 Duo 2.16 GHz, ATI RadeonX1600 GPU and 3 GB of 667 MHz DDR2 SDRAM. Perception is linear in the number of non-filtered image points for the depth image; (usually around 30 FPS). Octree segmentation is linear with the number of voxels within a combined point cloud model. Block grouping performance largely depends on the number of bins and how well they are grouped. Usually it takes around 6.5 seconds. Assembly grouping performance depends on the number of blocks; for the test models it usually takes around 6 seconds. If the knowledge base is preloaded execution time can be reduced by approximately 2.5 seconds for each of the last two steps.

7 Conclusions and Future Research

We have shown DUPLOG, a method for acquiring a model of Duplo block assemblies from raw image data. Our method achieves a high accuracy of complete building interpretation using an effective (and efficient) combination of state-of-the-art computer vision techniques such as point clouds and octree representations, and logic-based techniques such as probabilistic Prolog and logical constraints. We have shown how logically represented high-level knowledge can be employed to interpret features derived from images. Despite other efforts in logic-based computer vision, to the best of our knowledge, our combination of octree representations and probabilistic logic (in the context of a Duplo domain) is novel. Duplo blocks provide an interesting, regular domain. However, our four-step procedure is applicable beyond this domain: basically any structured object recognition system can be based upon it.

A prominent future direction are real-time interpretations, incrementally while building. In the vision part, our calibration phase could be automated completely, subsequent versions of the Kinect can be used, and all separate steps (corrections and filters) can be improved. For the logical part we can employ interpretation settings (MAP, abductive), other implementations (such as Markov logic networks), or use (hierarchical) layers of interpretation (e.g. to interpret higher level assembly concepts such as *bridge* or *tower*). Finally, we can extend our application domain to other structured objects, or to interpret complete visual scenes.

References

[1] L. Antanas, M van Otterlo, J. Mogrovejo Oramas, T. Tuytelaars, and L. De Raedt. There are plenty of places like home: Using relational representations in hierarchies for distance-based image understanding. *Neurocomputing*, 2013. In Press.

[2] H. Bunke and A. Sanfeliu. *Syntactic and Structural Pattern Recognition: Theory and Applications*. World Scientific Pub Co Inc, 1990.

[3] L. De Raedt. *Logical and Relational Learning*. Springer, 2008.

[4] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. Kinectfusion: Real-time 3D reconstruction and interaction using a moving depth camera. In *Proc of UIST'11*, 2011.

[5] C. H. Jackins and S. L. Tanimoto. Oct-trees and their use in representing three dimensional objects. *Computer Graphics and Image Processing*, 14(3):249–270, 1980.

[6] A. Miller, B. White, E. Charbonneau, Z. Kanzler, and J. J. Laviola Jr. Interactive 3D model acquisition and tracking of building block structures. *Tr. on Vis. and Cmp. Graph.*, 18(4):651–659, 2012.

[7] B. Moldovan, P. Moreno M. van Otterlo, J. Santos-Victor, and L. De Raedt. Learning relational affordance models for robots in multi-object manipulation tasks. In *International Conference on Robotics and Automation (ICRA)*, pages 4373–4378, 2012.

[8] B. Neumann and R. Möller. On scene interpretation with description logics. *Image and Vision Computing*, 26(1):82–101, January 2008.

[9] M. Peysakhov. Representation and evolution of lego-based assemblies. In *Proceedings of GECCO Graduate Student Workshop*, pages 297–300, 2000.

[10] A. J. Pinz, H. Bischof, W. G. Kropatsch, G. Schweighofer, Y. Haxhimusa, A. Opelt, and A. Ion. Representations for cognitive vision: A review of appearance-based, spatio-temporal, and graph-based approaches. *Electronic Letters on Computer Vision and Image Analysis*, 7:35–61, 2009.

[11] D. K. Prasad. Object detection in real images. *CoRR*, abs/1302.5189, 2013.

[12] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, 2010.