Inverse Algorithm Design of Floor Plans

Leveraging neural networks to convert floor plans into algorithmic designs

João David[1], Inês Pereira[2], Catarina Belém[3], António Leitão[4]
[1,2,4]INESC-ID/Instituto Superior Técnico [3]University of California Irvine
[1,2,4]{joaodavid|ines.pereira|antonio.menezes.leitao}@tecnico.ulisboa.pt [3]cbelem@uci.edu

Algorithmic design enables architects to create building designs using parametric algorithms, which can then be used in optimization workflows to enhance performance across various criteria. Unfortunately, algorithmic design has a steep learning curve, and it is difficult to convert existing designs to an algorithm-based representation. To address this problem, we propose inverse algorithmic design to automatically generate a building's algorithmic description from its digital representation. We focus on the common case of floor plans represented in raster images, and we explore recent computer vision techniques for floor plan interpretation. As part of our solution, we propose a new floor plan reconstruction method and a process to generate algorithmic design programs through metaprogramming. By leveraging prior knowledge of floor plan drafting conventions, we generate programs that are both parametric and easy to comprehend.

Keywords: *Computer Vision, Machine Learning, Algorithmic Design, Floor Plan Reconstruction, Metaprogramming, Parametric Design.*

INTRODUCTION

The architectural practice has come a long way since its beginning. Nowadays, Computer-Aided Drafting (CAD) and Building Information Modeling (BIM) tools are essential parts of architectural practice, enabling architects to draft their designs more accurately and efficiently. Even so, the traditional iterative design method suffers from the inherent difficulty of modifying designs once they have been defined (Jabi, 2013).

Algorithmic Design (AD) is a design paradigm that entails the use of algorithms to create digital representations of buildings (Caetano, Santos and Leitão, 2020). On one hand, AD is useful because the algorithmic steps can be defined in terms of parameters, allowing the architect to quickly try different variations of a given design just by adjusting the values of those parameters. Additionally, this parametric model of the design can be coupled with evaluation functions and optimization algorithms to automatically search for the best design solution regarding one or multiple performance aspects, such as daylight availability, energy consumption, or construction costs (Belém and Leitão, 2018). On the other hand, AD requires architects to learn how to depict buildings using algorithms, which is a considerable shift in perspective for those who come from a traditional background. Another significant problem, particularly for renovations, is that, currently, there is no easy way to convert existing work into an algorithmic representation: the algorithm would have to be written from scratch, a time-consuming task requiring specialized knowledge.

In this work, we propose Inverse Algorithmic Design (IAD), a process to generate an AD program from a digital representation of a building. We focus on the interpretation of floor plan images through Computer Vision (CV) techniques and the use of metaprogramming to generate the AD program. We also compare our approach to one based on the extraction of basic geometric entities from digital floor plan models.

RELATED WORK

IAD entails two main problems. One is the interpretation of floor plan images, and the other is the generation of equivalent algorithmic representations.

Concerning the first problem, traditional approaches usually make a set of assumptions about the floor plans' appearance and resort to low-level image processing techniques to identify the graphic elements. For example, Dosch et al (2000) separate thick from thin lines, assuming that only exterior walls are modeled by thick lines.

More recently, there has been a focus on Machine Learning (ML) techniques for recognizing floor plan images. Liu et al (2017) propose a system to convert raster floor plans to a vector format by using a neural network to segment the image and find junction points, and then formulate an Integer Linear Programming (ILP) problem to build a vector representation of the building. Their approach has three main setbacks: only axis-aligned walls are considered, wall thickness is discarded, and the formulated ILPs may be unfeasible. Lv et al (2021) propose a method for residential floor plan recognition and 3D model reconstruction. Unlike the previous work, (1.) vectorization is performed by an iterative process using segmented rooms, and (2.) the authors propose a method to recover wall thickness after vectorizing the results.

Other investigations try to improve the segmentation models that are crucial for these vectorization systems. Dodge, Xu and Stenger (2017) explore the use of fully convolutional networks with different strides for wall segmentation. Zeng et al (2019) propose an attention-inspired architecture for floor plan images with direction-aware convolutional kernels. Zhang et al (2020) improve on the results by making the direction-aware kernels learnable and by introducing an adversarial module through a generative adversarial network discriminator. Knechtel et al (2024) proposes a graph-convolutional network to predict room boundaries and their semantic labels.

There is also a large body of research on the use of ML for building design reconstruction from point clouds instead of architectural drawings. However, as Schönfelder et al (2023) argue, architectural drawings are more valuable for accurate building reconstruction because they provide additional information beyond what is visible in point clouds. In their systematic review, authors further emphasize the need for ML-based studies to not only consider architectural drawings but also to focus on modeling intricate components, since they significantly contribute to the overall building performance.

Regarding the second main problem of IAD, i.e., the generation of an equivalent AD representation, most research in this field addresses the inference of generative grammar-based systems, such as parametric L-systems (Šťava et al, 2010), probabilistic grammars (Talton et al, 2012), shape grammars (Bokeloh, Wand and Seidel, 2010), and set grammars (Wu et al, 2013). Many of the strategies of grammar inference involve shape decomposition, based on similarities of the elements of the input dataset, including symmetry detection. However, these approaches were only tested on designs with repeating elements and simple geometry that were far from real-world complex designs.

The approach proposed by Leitão and Garcia (2022) explores the design information stored in CAD applications to quickly generate a low-level non-parametric AD program that is isomorphic to the digital design represented in the CAD tool. This program is then mechanically transformed under the guidance of a designer/programmer to increase its legibility and parametricity.

Unfortunately, this second step still requires a considerable amount of human effort, making it an obstacle to its widespread adoption.

INVERSE ALGORITHMIC DESIGN

This section introduces IAD, our proposal for generating an AD program from a building's digital representation (see figure 1). Our solution is based on structured floor plan recognition (Liu et al, 2017; Lv et al, 2021) and consists of two main components: (1.) **Floor plan recognition**, which detects and structures key elements in the input image; and (2.) **Floor plan reconstruction.** which converts this structured output into an AD format.

Recognition

Our recognition process involves two sequential steps: segmentation, to classify image pixels based on the architectural elements they represent, and vectorization, to convert the pixel-based segmentation into structured data.

Vectorization. Our method combines the junction-based approach from Liu et al (2017) and the iterative vectorization from Lv et al (2021). As such, we require pixel-wise segmentation on the input image (with walls, rooms, doors, etc.) and a prediction of the position of the junction points. Unlike Liu et al (2017), we discard semantics regarding the junction types, allowing us to reconstruct diagonal walls.

Similarly, we start by extracting the junction points through heat map regression, which are then filtered by retaining only the peak values. We then go over all combinations of pairs of points and, from the closest points to the farthest ones, we check whether there is a large portion of wall pixels in a thin section between the two points. We consider that a wall exists between these two points if the fraction of pixels in the direction of the wall axis is greater than an empirically defined threshold τ_w.

By detecting walls based on the closest pairs of points first, we prevent walls from overlapping junction points, which we consider the foundational elements of the building's structure. Additionally, for each point, we store the angle of the candidate walls that have been tested. As such, walls are never placed over junction points, as the shortest wall must be tested first, invalidating any possible longest wall in the same direction. As the extracted points might not be perfectly aligned, we define an angle tolerance parameter τ_t. We also record the angle of placed walls to avoid placing two walls that make a tight angle according to a threshold τ_p. We set τ_w=0.75, τ_p=67.5°, and τ_t=5° from empirical observation.

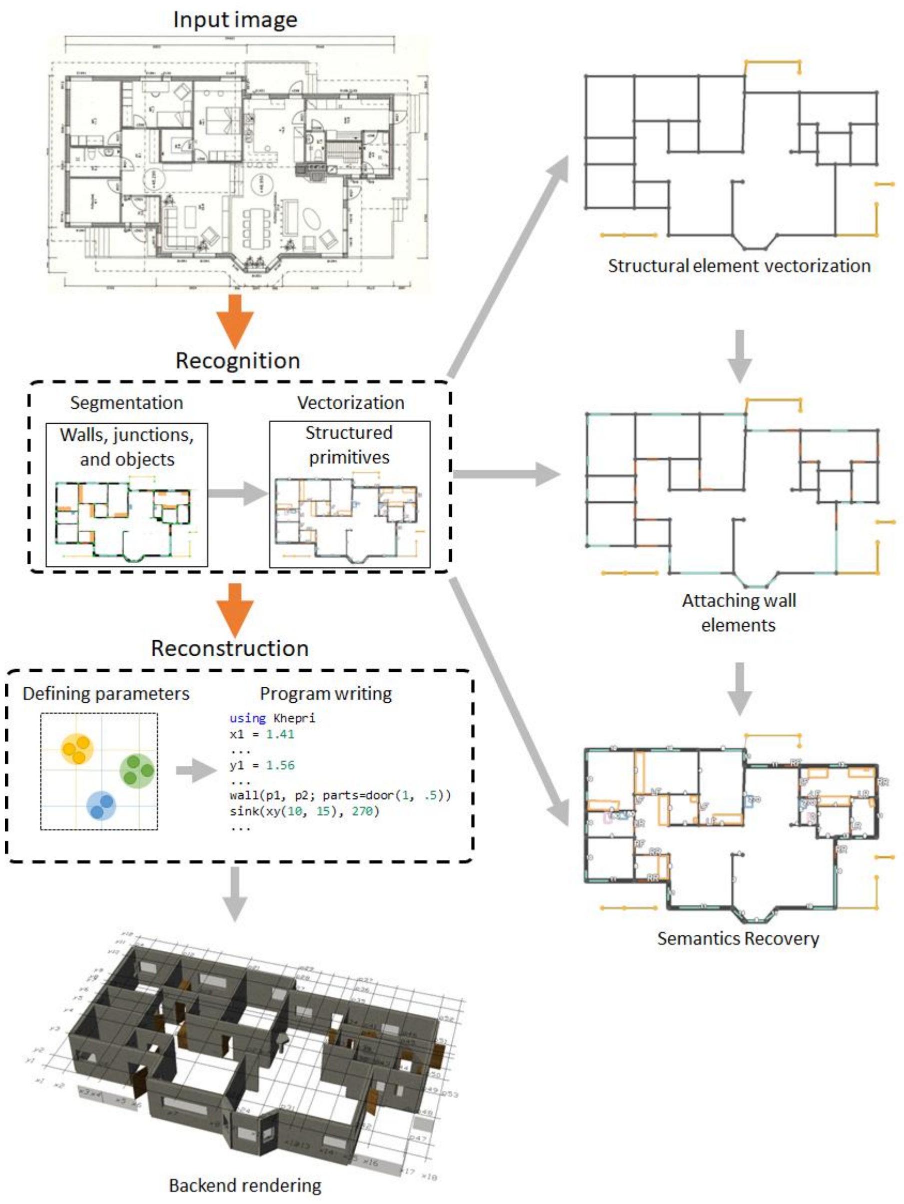
Unlike Lv et al's (2021) work, our method is not capable of detecting curved walls, whereas theirs approximates them with small segments. This is not a concern, as curved walls are rare, and approximating them with small segments would only increase the number of points and parameters in the reconstructed floor plan.

As for recovering wall thickness, we follow the approach proposed by Lv et al (2021). For each identified wall, an optimization process is run to find which wall thickness is a better match to the segmented image. The authors used Intersection Over Union (IoU) as a fitness function; however, in our experiments, IoU performs poorly on small walls, as increasing wall thickness often boosts the metric by capturing surrounding walls. Instead, we use weighted subtraction between the number of intersecting pixels (true positives—TP) and the generated pixels that do not intersect the segmentation results (false positives—FP):

$$\text{fitness} = \lambda_1 TP - \lambda_2 FP.$$

We set λ_1=1 and λ_2=1.1 so that we only grow wall thickness as long as there are more true positives than false positives, giving slightly more importance to false positives when the amount of both is very similar.

Segmentation. Our vectorization algorithm requires a model capable of (1.) segmenting walls, rooms, and structural and decorative elements, and (2.) predicting junction points.



```
using Khepri
x1 = 1.41
...
y1 = 1.56
...
wall(p1, p2; parts=door(1, .5))
sink(xy(10, 15), 270)
...
```

Figure 1
Overview of our IAD proposal. The recognition phase is composed of segmentation, vectorization, and semantics recovery. The reconstruction phase defines parameters from shared values and writes the final AD program.

We use the model proposed by Kalervo et al (2019), which employs the same ResNet-152-based architecture (He et al, 2016) used in Raster-to-Vector (Liu et al, 2017), and we train it on the CubiCasa5K dataset (Kalervo et al, 2019). This dataset supports three tasks: room type prediction (including walls), icon prediction, and junction point prediction. The model's output consists of 13+4+4 heat maps (wall, icon, and wall element junctions), plus 12 segmentation maps for room types and 11 for icons. Then, to convert junction-point predictions into a list of candidate points, we must set a confidence threshold *t*.

As our vectorization method is agnostic to junction types, we merge all 13 heat maps into a single one by taking their maximum values and extract the junction points on the combined map. Then, we followed the evaluation process in Raster-to-Vector on the created heat map and ignored the ground truth junction types. We plotted the precision and recall for threshold values t={0.05,0.1,0.2,...,1} and chose the best point based on the F1-score. We chose t=0.2 with an F1-score of 79.73%, corresponding to 78.14% recall and 81.38 % precision.

Symbol classification

The next step in our IAD pipeline is enriching the identified symbols with additional information. To recover finer-grained information about symbols, we apply multi-class classifiers to each identified symbol. We start with the classification of door types and their orientation, and then we explore how to recover information regarding the orientation of other symbols.

Doors. David and Leitão (2023) proposes a method to classify doors in floor plan images: they classify them based on its opening direction (forward or reverse) and the position of its hinges (left or right), yielding four possible classifications for a normal swing door. The same rationale is applied to double doors and doors that open in both directions. For the remaining door types, no information regarding their orientation is added.

We then integrate this door classifier into the general floor plan recognition process. We find the contours of the segmented wall openings and approximate them with a rectangle. The start and endpoints are extracted from the rectangles by considering their longest sides. As doors are embedded in walls and, at this point, their orientation might not match perfectly, we first embed them into their closest wall and use that to correct the doors' orientation. Then, the points are ordered to maintain consistency with the annotated images, and the rectangle is expanded so that it can fit any type of door, including the arc. Finally, the rectangle is used to extract a cropped area from the original image, and the trained model classifies it, yielding the door type.

Toilets and sinks. In this section, we focus on recovering the orientation of other symbols, focusing on toilets and sinks. The same approach can be applied to other symbols, like furniture, just by training the model with an adequate dataset. We simply extract a rotated bounding box from the segmentation contours, and we recover their orientation by using a value between 0° and 359° to denote their angle.

For the toilets, we create a dataset using a method similar to the one used for doors. By analyzing the annotations in the CubiCasa5K dataset, we note that toilet symbols include a bounding box; thus, we extract the bounding boxes' points in the same order so that all images have the same orientation. The final images are similar, containing the symbol with its centroid at the center and the same orientation. The images are cropped into a square shape, considering the longest side with an additional margin for more context. The generated dataset contains 6894 images. Figure 2 shows examples of images extracted from these datasets.

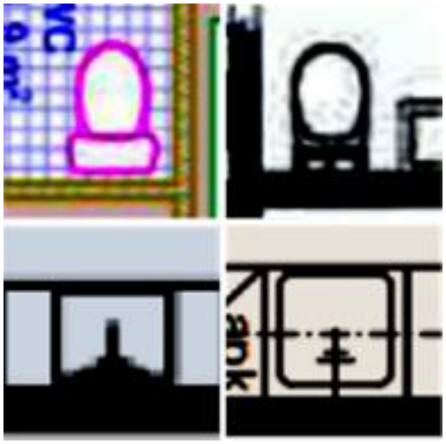


Figure 2
Two images from the toilets dataset (top) and two from the sinks dataset (bottom).

To label the data, we follow the classification approach in RotNet, i.e., the training images are rotated by a random number of degrees *x* that we use as the class for the generated example. This method is, however, inappropriate for the validation and test datasets, as those would become non-deterministic. For these datasets, we generate all possible combinations of *(image, rotation)*. To speed up training, we randomly pick a sample of 1000 combinations to use across all epochs, rather than using all 4248 images after splitting. As for transformations, we use random cropping, color jittering, and horizontal flipping.

We simplify the rotation classification by focusing on 45 ° rotations instead of 1-degree rotations, leading to 8 classes instead of 360. This assumption also reflects the vast majority of real buildings, and it can be further refined using the surrounding context, as toilets must be attached to walls. We then train a ConvNeXt-Tiny using cross-entropy loss and the Adam optimizer until convergence.

We extracted the sinks likewise, generating 12983 images. Unlike toilet symbols, sinks contain additional information indicating the type of sink. We noticed that sinks installed in corners were identified with a distinct "CornerSink" class, and their orientation did not match the orientation of the sinks of the remaining types. Out of the 12983 images, only 47 are labeled as corner sinks.

We tackle the data balancing problem by following an oversampling approach so that the minority class makes up 40% of the training data. We chose this fraction as this class contains 9 possible orientations, while the other classes correspond to a single orientation. We use the same ConvNeXt-Tiny with the same 45° granularity for classification and add one additional class for corner sinks. As previously, we use a smaller sample of 1000 examples during validation and cross-entropy loss, and train the model until the validation loss stops improving.

Algorithmic design generation

In this section, we cover our method of converting the previously extracted elements (i.e., walls, doors, windows, toilets, and sinks) into an executable AD program.

Reconstruction and integration. Khepri (Leitão, Castelo-Branco and Santos, 2019) is an AD tool that allows one to create designs through programs written in the Julia programming language. One of its main advantages is that the same program is portable across different tools, not only CAD and BIM tools but also game engines and analysis tools. We use Khepri to easily map the extracted information about walls and symbols to code. As our final program is written in Julia, we use the same language to implement our IAD system, thus enabling the use of Julia's metaprogramming capabilities.
Having introduced how we integrate the recognition component and the reconstruction component, we now cover how the reconstruction is processed. The system receives a path to an image as input, returning the detected elements. We focus mainly on parameterizing the structural elements, as those are the most important for potential building optimization. We start by analyzing which parameters can be shared between elements.

As it is common for walls to share lines, even when not adjacent, we start by extracting all junction points and their respective *x* and *y* components. Then, we cluster these components together by defining a maximum distance for the values within each cluster.

To create these clusters, we use a hierarchical clustering algorithm with the complete-linkage criterion, as it ensures that points inside a cluster do not have a distance greater than an empirically defined threshold. The points are replaced by tuples containing the cluster index for each component, and the mean value of each cluster is used as the real value for the assigned components. Figure 3 illustrates the clustering process when applied to points.

Besides coordinates, we also cluster other often-used values. As architectural drawings usually use conventionalized wall thickness for exterior walls and a different thickness for interior walls, we create two clusters for wall thickness. We also cluster door and window lengths, but in the case of these two elements, we use a length-difference threshold rather than a predefined number of clusters.

One important aspect when defining these thresholds is that they should be meaningful, which is hard to achieve with raw pixel numbers. We use the clustering process to estimate the building's scale based on thickness. Because exterior walls are usually more consistent across buildings, we pick the second wall thickness cluster (the thickest walls) and calculate the median pixel thickness.

These steps allow us to gather all the necessary information to generate an AD program capable of accurately describing a building. To achieve this, we leverage Julia's metaprogramming capabilities.

Generated code. The generated program consists of two main files: (1.) an auxiliary file used by all generated programs that contains data structures and utility functions to make the generated code more readable, and (2.) the file with the program generated for the specific input floor plan.

We also generate lines and text labels to aid with visualization. For each x value, we draw a vertical line as well as its corresponding variable name. The same process is applied to y values, and text labels are also placed near each junction point, identifying their corresponding variable. This allows the user to easily identify where each variable is being used, facilitating its manipulation. Finally, we run a code formatter to improve the readability of the program by ensuring consistency with some common styling conventions.

The automatic labeling step is important for the next phase of the process, where a designer/programmer analyses the generated program and refactors it to make it more abstract and more parametric. This phase explores the semi-automatic refactoring techniques presented by Leitão and Garcia (2022) to improve the parametricity of the generated program without introducing bugs. The resulting program can then be used to generate the design variations needed for the optimization of the design.

EVALUATION

Here, we present an evaluation of our solution, starting with a quantitative evaluation of its components, showing examples generated by our program, and ending with a comparison between our method and other approaches.

Our segmentation model is the same as the one provided by the CubiCasa5K authors, but we got slightly different results. The overall accuracy and mean IoU for room segmentation are 82.5% and 57.3%, respectively. For the icons, these metrics are 97.5% and 55%.

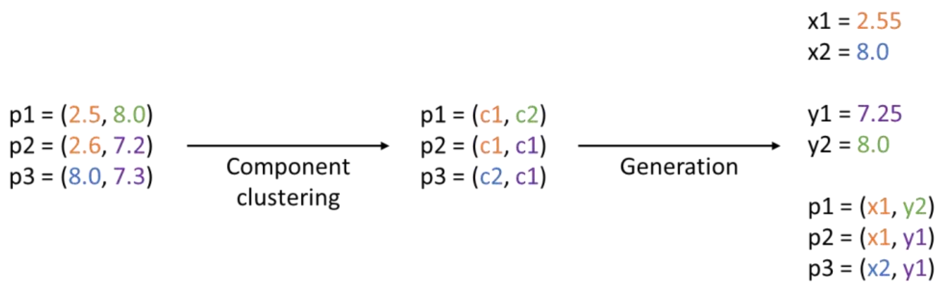


Figure 3
The individual components are first clustered, and an indexed description of each point is created. The points are then generated by assigning the clusters' mean to their values, and the points use the clusters' indices directly. Additionally, p1 and p2 are switched on the last step to facilitate reading.

We analyzed junction points without considering their types, as our vectorization method does not depend on them. The results on the test set are consistent with those on the validation set, maxing out the F1-score at 76.7% with a detection threshold t=0.2.

For toilet classification, our model achieves 98.9% overall accuracy on the test set. As the data is balanced, this value serves as a good indicator of the model's performance. For the sink dataset, the model achieves 98.5% overall accuracy and correctly classifies most corner sinks (minority class) without affecting other classes. The model achieves 95.8% accuracy for corner sinks and an average of 98.6% for the remaining ones.

In Figure 4, we show a few generated examples, all from the CubiCasa5K test set. All examples were generated with the assumption that the exterior walls are 35cm thick. We use 30cm for the maximum cluster distance. No manual intervention was performed on the resulting programs and 3D models.

We also compare the proposed solution with that of a non-CV-based one, namely the Reverse Algorithmic Design (RAD) (Leitão and Garcia, 2022), which directly explores the vectorial information contained in CAD applications. The RAD approach consists of two main steps: (1.) **Extraction**, which uses metaprogramming to generate a low-level, AD program from a CAD design, and (2.) **Refactoring**, where a human designer-programmer transforms the program to improve its abstraction level and readability. The initial program is not only very low-level but also non-parametric, making the second step essential for usability. This step is supported by a refactoring tool to avoid introducing bugs during the transformation and a traceability tool to map program components to their corresponding design elements. However, refactoring is still a human-guided and time-consuming process, often taking hours even for simple floor plans.

CONCLUSIONS AND FUTURE WORK

AD is an intrinsically parametric design paradigm that enables iterative design and optimization. However, it faces several challenges such as a steep learning curve, a low adoption rate, and difficulty in converting existing projects. To address these issues, we propose IAD, a process that automatically generates an AD program from floor plan images using CV techniques and metaprogramming.

The generated program is parametric and tailored to a portable AD tool—Khepri. This means that the same algorithmic description can be used to generate the model in different tools, including CAD and BIM tools. Among the parameterizable program elements, we include (1.) the position and axis of the walls, which can be used to change the size of the rooms and corridors, (2.) the size, direction, and type of the

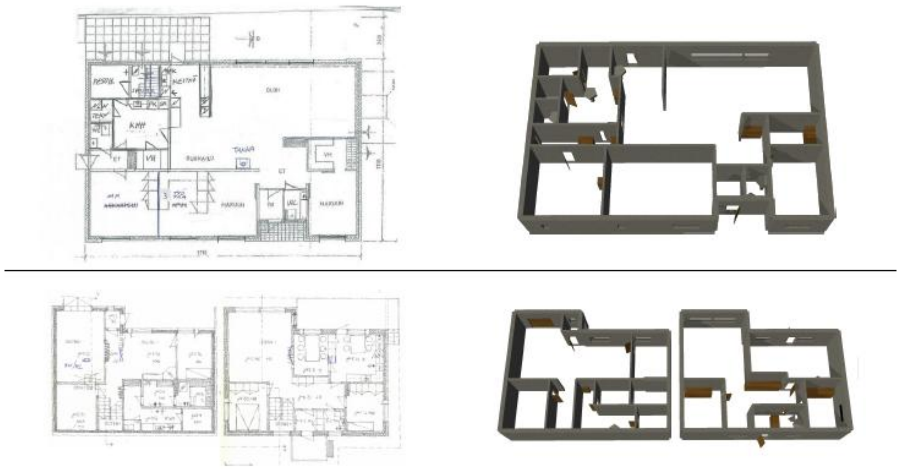


Figure 4
Input floor plans (on the left) and their corresponding 3D models created by our system (on the right).

doors, and (3.) the position and size of the windows. These parameters can then be used to optimize the design, such as by maximizing natural light in the rooms or minimizing the construction materials.

Although our proposal advances the state of the art, there is still room for improvement. One obvious move would be to improve the models' performance by finding better ML architectures or training with more data. Our method assumes the input floor plan is a raster image. However, floor plans are often available in vector format. Therefore, we plan to extend our method to support vector floor plans, which can improve measurement accuracy and simplify element extraction. We also note that our vectorization method cannot recognize curved walls or other more complex geometries. Recognizing these would be an interesting addition. Curved walls, for instance, could be represented by parametric curves such as arcs and Bézier curves.

DATA AVAILABILITY

The data and code are available in the following repositories:

- https://github.com/joaocmd/CubiCasa5k
- https://github.com/joaocmd/Inverse-Algorithmic-Design

ACKNOWLEDGMENTS

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (DOI: 10.54499/UIDB/50021/2020) and by the PhD grant under contract with FCT, reference 2021.06302.BD (DOI: 10.54499/2021.06302.BD).

REFERENCES

Belém, C. and Leitão, A. (2018). 'From design to optimized design: An algorithmic-based approach', in Kępczyńska-Walczak, A. and Bialkowski, S. (eds.) *Computing for a Better Tomorrow - Proceedings of the 36th eCAADe Conference*, 17-21 September, Lodz, Poland, pp. 549-558.

Bokeloh, M., Wand, M. and Seidel, H.P. (2010). 'A connection between partial symmetry and inverse procedural modeling', *ACM Transactions on Graphics*, 29(4), pp. 1-10.

Caetano, I., Santos, L. and Leitão, A. (2020). 'Computational design in architecture: Defining parametric, generative, and algorithmic design', *Frontiers of Architectural Research,* 9(2), pp. 287-300.

David, J. and Leitão, A. (2023). 'Getting a handle on floor plan analysis: Door classification in floor plans and a survey on existing datasets', *Architecture & Planning Journal*, 28(3), art. 6.

Dodge, S., Xu, J. and Stenger, B. (2017). 'Parsing floor plan images', in *Proceedings of the 15th IAPR MVA Conference*, 8-12 May 2017, Nagoya, Japan, pp. 358-361.

Dosch, P., Tombre, K., Ah-Soon, C. and Masini, G. (2000). 'A complete system for the analysis of architectural drawings', *IJDAR*, 3(2), pp. 102-116.

He, K., Zhang, X., Ren, S. and Sun, J. (2016). 'Deep residual learning for image recognition', in *Proceedings of the IEEE CVPR Conference*, 27-30 June 2016, Las Vegas, USA. pp. 770-778.

Jabi, W. (2013). *Parametric design for Architecture*. 1st edn. United Kingdom: Laurence King Publishing.

Knechtel, J., Rottmann, P., Haunert, J. H. and Dehbi, Y. (2024). 'Semantic floorplan segmentation using self-constructing graph networks'. *Automation in Construction*, 166(10), pp. 105649.

Kalervo, A., Ylioinas, J., Haikio, M., Karhu, A. and Kannala, J. (2019). 'CubiCasa5k: A dataset and an improved multi-task model for floorplan image analysis', in Felsberg, M., Forssén, PE., Sintorn, I.M. and Unger, J. (eds.) *Image Analysis - Proceedings of the 21st SCIA*, 11-13 June 2019, Norrköping, Sweden, pp. 28-40.

Leitão, A., Castelo-Branco, R. and Santos, G. (2019). 'Game of renders: The use of game engines for architectural visualization', in Haeusler, M.H., Schnabel, M.A. and Fukuda, T. (eds.) *Intelligent & Informed - Proceedings of the 24th CAADRIA*, 15-18 April 2019, Wellington, New Zealand, pp. 655-664.

Leitão, A. and Garcia, S. (2022). 'Reverse algorithmic design', in Gero, J.S. (ed.) *Design Computing and Cognition'20*, 14-16 December 2020. Atlanta, USA, pp. 317-328.

Liu, C., Wu, J., Kohli, P. and Furukawa, Y. (2017). 'Raster-to-vector: Revisiting floorplan transformation', in *Proceedings of the IEEE ICCV Conference*, 22-29 October 2017, Venice, Italy, pp. 2195-2203.

Liu, Z., Mao, H., Wu, C.Y., Feichtenhofer, C., Darrell, T. and Xie, S. (2022). 'A ConvNet for the 2020s', in *Proceedings of the IEEE/CVF CVPR Conference*, 18-24 June 2022, New Orleans, USA, pp. 11976-11986.

Lv, X., Zhao, S., Yu, X. and Zhao, B. (2021). 'Residential floor plan recognition and reconstruction', in *Proceedings of the IEEE/CVF CVPR Conference*, 20-25 June 2021, Nashville, USA, pp. 16717-16726.

Schönfelder, P., Aziz, A., Faltin, B. and König, M. (2023). 'Automating the retrospective generation of As-is BIM models using machine learning', *Automation in Construction,* 152(8), pp. 104937.

Št'ava, O., Beneš, B., Měch, R., Aliaga, G. and Kriš tof, P. (2010) 'Inverse procedural modeling by automatic generation of L-systems', *Computer graphics forum*, 29(2), pp. 665-674.

Talton, J., Yang, L., Kumar, R., Lim, M., Goodman, N. and Měch, R. (2012). 'Learning design patterns with Bayesian grammar induction', in *Proceedings of the 25th ACM symposium on UIST*, 7-10 October 2012, Cambridge, USA, pp. 63-74.

Wu, F., Yan, D.M., Dong, W., Zhang, X. and Wonka, P. (2013) 'Inverse procedural modeling of facade layouts', in *arXiv:1308.0419*.

Zeng, Z., Li, X., Yu, Y.K. and Fu, C.W. (2019). 'Deep floor plan recognition using a multi-task network with room-boundary-guided attention', in *Proceedings of the IEEE/CVF ICCV Conference*, 27 October - 02 November 2019, Seoul, Korea (South), pp. 9096-9104.

Zhang, Y., He, Y., Zhu, S. and Di, X. (2020). 'The direction-aware, learnable, additive kernels and the adversarial network for deep floor plan recognition', in *arXiv:2001.11194*.