# Vectorizing Building Blueprints

Weilian Song*, Mahsa Maleki Abyaneh, Mohammad Amin Shabani, and Yasutaka Furukawa

Simon Fraser University, BC, Canada

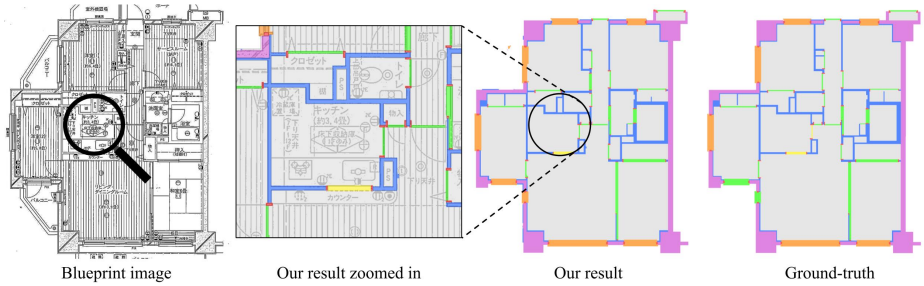


**Fig. 1.** This paper proposes a novel algorithm for vectorizing building blueprints with intricate architectural details. The left is an input raster blueprint image. Our vectorized blueprint is shown in the middle two columns (the left is a zoomed-in view). The right is the ground-truth.

**Abstract.** This paper proposes a novel vectorization algorithm for high-definition floorplans with construction-level intricate architectural details, namely a blueprint. A state-of-the-art floorplan vectorization algorithm starts by detecting corners, whose process does not scale to high-definition floorplans with thin interior walls, small door frames, and long exterior walls. Our approach 1) obtains rough semantic segmentation by running off-the-shelf segmentation algorithms; 2) learning to infer missing smaller architectural components; 3) adding the missing components by a refinement generative adversarial network; and 4) simplifying the segmentation boundaries by heuristics. We have created a vectorized blueprint database consisting of 200 production scanned blueprint images. Qualitative and quantitative evaluations demonstrate the effectiveness of the approach, making significant boost in standard vectorization metrics over the current state-of-the-art and baseline methods. We will share our code at https://github.com/weiliansong/blueprint-vectorizer.

**Keywords:** Vectorization · Blueprint · Segmentation.

## 1 Introduction

Blueprints are technical drawings of buildings, conveying rich architectural and engineering information for building maintenance, assessing building code compliance, and remodeling. Unfortunately, blueprints are often stored as scanned

* Corresponding author, weilians@sfu.ca


**Fig. 2.** Sample floorplans from the R2V [11] dataset (left two) in comparison to our dataset (right two).

raster images, where a professional architect spends hours manually converting to a vector format for down-stream applications. Automated blueprint vectorization will have tremendous impacts on real-estate or construction industries, significantly reducing the amount of human resources in converting hundreds of thousands of blueprints ever scanned and stored as raster images.

Floorplan vectorization has a long history in the domain of document scanning [3,9]. A classical approach relies on heuristics and was never robust enough at a production level. With the advent of deep learning, Liu *et al.*made a breakthrough in combining convolutional neural networks for corner detection and binary integer programming for their connection inference [11]. Their system achieves more than 90% precision and recall for the task of vectorizing consumer-grade floorplan images in the real-estate industry.

In this paper, we propose a novel vectorization algorithm for building blueprints with intricate architectural details. Our system outputs the floorplan in vector-graphics representation, and keeps intermediate representations as raster segmentation. A standard vectorization process starts by corner detection, which fails severely on complex blueprints. Our approach 1) starts by region-detection (which involves higher level primitives and is more robust) by an instance segmentation technique; 2) detects missing or extraneous architectural components by topological reasoning; 3) adds or removes components by a refinement generative adversarial network; and 4) further refines the boundaries by heurisics.

We have annotated 200 raster blueprints as vector-graphics images. We compare against the current state-of-the-art and baseline methods, based on the standard metrics as well as a new one focusing on the topological correctness. The proposed approach makes significant improvements over existing methods.

## 2 Related Works

We review related literature, namely, primitive detection, floorplan vectorization, and structured reconstruction.

**Primitive detection**: A floorplan is a 2D planar graph, consisting of three levels of geometric primitives: 0-dimensional corners, 1-dimensional edges, and 2-dimensional regions. Primitive detection is a crucial step for floorplan construction, where convolutional neural networks (CNNs) have proven effective. Fully convolutional architecture produces a corner confidence image, where non maximum suppression finds corners [15,4]. The same architecture produces an "edge confidence image", where a pixel along an edge has a high value. Direct detection of edge or region primitives is possible by object detection networks such as faster-RCNN [16].

Standard instance segmentation techniques such as Mask-RCNN [8] and metric learning [6] yield primitive segmentation. However, blueprints contain many extremely thin and elongated regions, where these techniques perform poorly. We divide an input image into smaller patches, perform metric learning locally per patch, and merge results.

**Floorplan vectorization**: Liu *et al.* [11] presented the first successful floorplan vectorization approach by combining CNNs for corner detection and Integer Programming for edge/region inference. However, their approach is unsuitable for our problem due to two key differences in the data. First, their data are consumer-grade floorplans with much simpler graph structures in comparison to our blueprints. Second, their input are digitally rasterized images (e.g., a jpeg image), while our input are optically scanned images from printed papers, exhibiting severe noise and distortions. Some sample floorplans from Liu *et al.* [11] are shown in Figure 2. Our approach is to 1) only utilize region detection (i.e., no corner nor edge detection, which are less robust) and 2) learn to add or remove tiny architectural components, which are the hardest to detect, by topological reasoning.

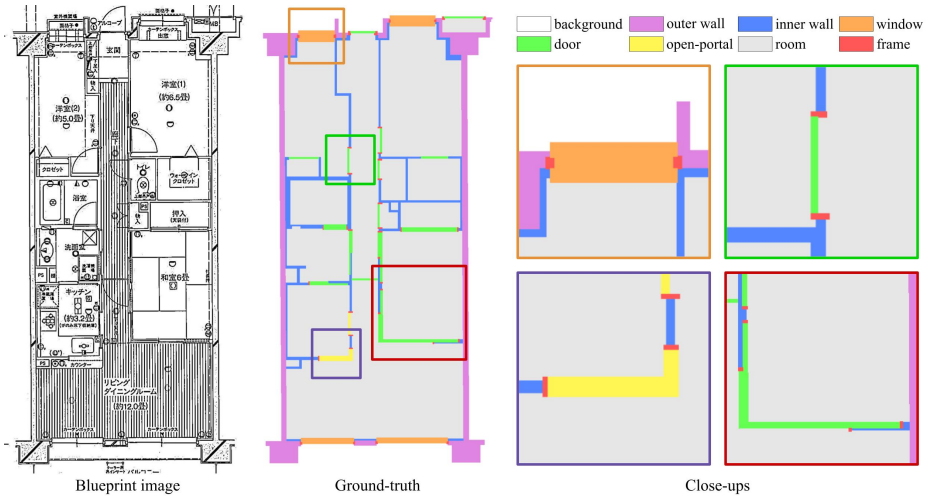


**Fig. 3.** A sample annotated blueprint with close-ups.

**Table 1.** Statistics of our dataset, consisting of 200 blueprint images. All averages are rounded to the nearest integer.

| | Image | | Primitives | | | Regions by type | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Height | Width | Corners | Edges | Regions | Outer | Inner | Window | Door | Portal | Room | Frame |
| Total | N/A | N/A | 123,230 | 139,808 | 17,077 | 1,384 | 2,111 | 1019 | 3,203 | 172 | 3,756 | 5,432 |
| Average | 1,279 | 906 | 616 | 699 | 85 | 7 | 11 | 5 | 16 | 1 | 19 | 27 |

**Structured reconstruction**: Structured reconstruction [12] seeks to turn raw sensor data (e.g., images or point-clouds) into structured geometry representation such as CAD models. The challenge lies in the noisy input sensor data. A general rule of thumb is to utilize higher level primitive detection (i.e., regions) instead of corner/edge detection, which is less robust [5,13]. Our approach also borrows this idea.

The state-of-the-art structured reconstruction algorithms handle outdoor architecture [19,13] or commercial floorplans [12,5], which are much simpler than the building blueprints. Our approach learns topological reasoning to handle tiny architectural components as described above.

## 3 Blueprint Vectorization Dataset

We collected 200 building blueprints from a production pipeline. Following the annotation protocol by an existing floorplan vectorization paper [11], we use the VIA annotator [7] to annotate a planar graph and associate an architectural component type to each region. There are eight architectural component types: background, outer wall, inner wall, windows, doors, open-portals, rooms, and frames. Table 1 shows various statistics of our dataset, and Figure 3 illustrates a sample blueprint along with its annotation. We do not have access to previous datasets [11] to compare statistics, but ours is many times more complex.

To expand upon our semantic types: open-portals denote an opening between rooms without a physical door. Frames are door trims that often surround doors within our dataset. From the top-down view, they appear as small rectangles on the ends of a door (small red rectangles in Figure 3). We make distinction between outer (exterior) and inner (interior) walls, which can be adjacent to each other for architectural reasons.

## 4 Blueprint Vectorization Algorithm

A high-level view of our system is shown in Figure 4. Our blueprint vectorization consists of four steps: 1) instance segmentation and type classification; 2) frame connectivity inference; 3) frame correction; and 4) ad-hoc boundary simplification. We first explain our intermediate data representation, then provide details of each step.

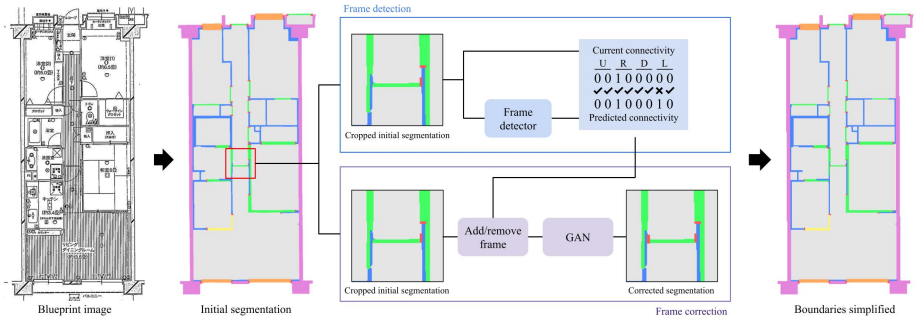


**Fig. 4.** System overview. After obtaining our initial segmentation through instance segmentation followed by type classification, we detect and correct missing/extraneous building-frames. Segment boundaries are simplified with heuristics to produce the final result.

### 4.1 Blueprint representation

While the goal is to reconstruct a blueprint as a planar graph, our algorithm also uses a pixel-wise semantic segmentation image as an intermediate representation, which is an 8-channel image as there exist 8 component types. Note that a semantic segmentation image is instance-unaware, and cannot represent two instances of the same type sharing a boundary. However, such cases are rare and we merge such instances into a single segment for simplicity.

A planar graph is converted to a semantic segmentation image by simple rasterization. A semantic segmentation image is, in turn, converted to a graph representation by tracing the segmentation boundaries while assuming a Manhattan world (i.e., edges are either horizontal or vertical). For example, a single-pixel segmentation is converted to a square with four corners in the vector representation.

### 4.2 Instance segmentation and type classification

**Instance segmentation**: Blueprint images need to be in high resolution to retain intricate geometric structures. We found that standard instance segmentation techniques such as Mask-RCNN [8] or metric learning [6] do not work well. Semantic segmentation [18,10] is also an option (given our semantic segmentation representation), but it does not work well due to the data-imbalance (i.e., the open-portal is the rare type as shown in Table 1).

We divide a scanned blueprint image into a set of overlapping "crops", each of which is a $256 \times 256$ grayscale image and overlaps with the other crops by 32 pixels. [1] Since Mark-RCNN is not effective for thin structures, we use a metric-learning based approach [6].

[1] At training, we randomly pick a pixel and extract a patch around it.

First, a standard CNN converts a crop into an embedding of size $256 \times 256 \times 8$ (i.e., 8 dimensional embedding per pixel) with a discriminative loss function [6] (implementation borrowed from [1]), which optimizes pixel embedding from the same instance to be similar and vice-versa. We refer to the supplementary for the architectural specification.

Second, a mean-shift clustering algorithm extracts instances from the embedding (i.e., a python module MeanShift from the scikit-learn library with bin-seeding and the bandwidth parameter set to 2.0). We also classify an architectural component type for each instance (see the paragraph below), and merge results by simple heuristics: A pixel in an overlapping region has multiple segmentation results from multiple crops. We take the most frequent component type within a $3 \times 3$ window centered at the pixel in all the crops.

**Type classification**: We train a CNN-based classifier to assign component type to each instance in each crop. For each instance in a crop, we pass the blueprint image of the crop and the binary segmentation mask of the instance as a 4 channel image to a standard CNN-based encoder with a softmax cross-entropy loss. Again, see the supplementary for the full architectural specification. To handle a large instance, we uniformly sample 20 pixel locations from the binary instance mask and use them as centers to extract crops. At test time, we take the average probability distribution from all the crops and pick the type with the highest probability.

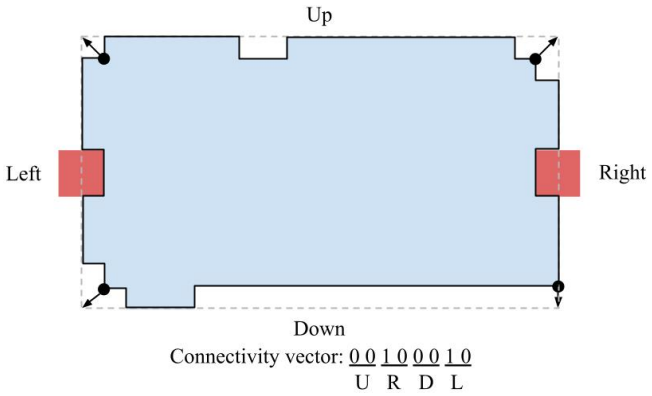


**Fig. 5.** Figure illustrating the heuristic used to determine the four sides of a segment. Dashed lines indicate the bounding box of a segment. Four disks indicate the vertices closest to the bounding box corners. We then trace the perimeter between the vertices to obtain the four sides.

### 4.3 Frame detection module

Small architectural components tend to be missed by the instance segmentation technique in the first step. We train a classifier that infers the correct "connec-

tivity" of frames. Precisely, we define the connectivity of a segment to be a set of neighboring segments.

To encode the frame connectivity, we first apply a heuristic to determine the segment's four sides. Then, assuming a maximum of two frames on each side[2], we use an 8-bit binary vector to denote the presence of frames on all four sides. Figure 5 illustrates the heuristic. The top-left, top-right, bottom-left, and bottom-right corners of the instance are first determined, which are defined as the four vertices closest to the instance bounding box. The four sides are then obtained as boundary edges in between two corners, and each frame is attached to one side only.

With our representation, we train a classifier to predict a segment's ground-truth frame connectivity. The classifier takes as input three masks: a one-hot encoded semantic segmentation mask around the segment, the corresponding gray-scale blueprint crop, and a binary mask highlighting the segment. The network directly outputs the 8-bit vector, and is optimized with the standard binary cross-entropy loss. We use a mix of ground-truth and predicted data to generate training input-label pairs, where each predicted instance is assigned the frame vector of the corresponding ground-truth instance. See supplementary material for more details on training data generation.

### 4.4 Frame correction module

Once we detect a discrepancy in the current and predicted frame arrangement, we employ a generative adversarial network (GAN) to locally correct the segmentation.

The main input to the generator is a graph, where each node contains a noise vector, an instance mask for the target segment, the blueprint crop, a one-hot type encoding, and a binary indicator denoting the absence of a frame. The instance mask is trinary with values $[-1, 0, 1]$; $-1$ and 1 pixels are the input constraints, and 0-pixels indicate areas of uncertainty. The generator learns to expand input constraints to fill in gaps between instances, and generate missing frames in nodes when indicated. Before inference, for an extra frame, we simply remove the corresponding node from the graph. For a missing frame, we add a new node, compute its approximate centroid location, and paint a square zero-mask of $18 \times 18$ pixels around the centroid in the instance mask. For all other segments, we randomly (50% chance) hide their boundaries by dilating and eroding the instance mask, and marking the difference as 0-pixels. See Figure 6 for an example input of door with missing frame, and please refer to the supplementary material for heuristics on approximate centroid computation.

Our generator and discriminator designs are inspired by House-GAN++ [14]. They are both convolutional message passing networks (ConvMPN) [19], and considers an input blueprint crop as a graph of instances. For simplicity, we consider the graph to be fully-connected. The generator first encodes input masks down to a lower spatial resolution, then performs a series of four convolutional

[2] One window in our dataset does not follow this rule, which we ignore for simplicity.

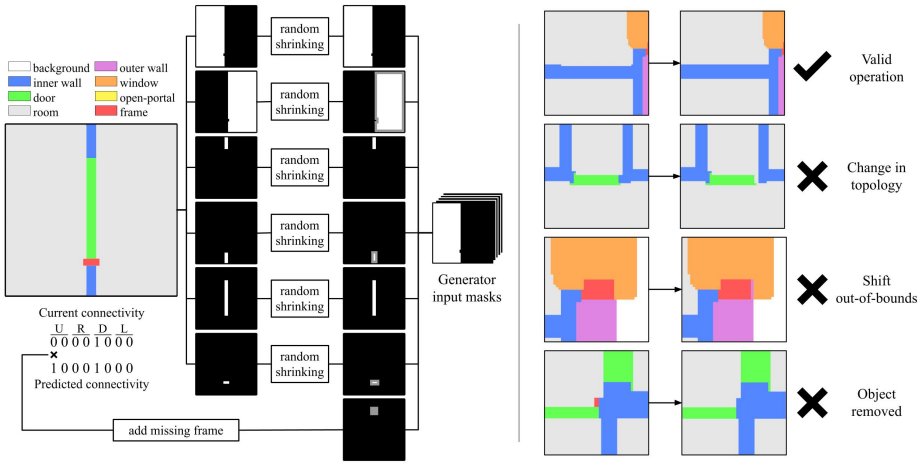


**Fig. 6.** Visualization of generator preprocessing routine (left) and example valid/invalid edge shift for heuristic simplification. (right)

message passing and up-sampling layers back to the input resolution. The discriminator performs a series of three convolutional message passing and down-sampling layers, followed by a special pooling and FC module to output a single value for the input graph. During training, we do self-supervised learning on the ground-truth data by randomly hiding frame instances. During testing, following House-GAN++ we do iterative refinement by running the generator a max of 50 steps or until convergence. Please see the supplementary for details.

### 4.5 Heuristic Simplification

Given our corrected segmentation, we use heuristics to simplify segment boundaries. Concretely, we take an edge shorter than 2 pixels (i.e., a reference edge) and consider neighboring edges that share one of its end-points. We find the neighboring edge that is perpendicular and closest to the reference. We shift the reference perpendicularly until the closest neighboring edge gets contracted. We perform this operation for each edge shorter than 2 pixels, unless one of the following three conditions occur (see Figure 6).

**Change in topology**: If an edge shift changes the segment connectivity (e.g., a door being disconnected from a wall).

**Shift out-of-bounds**: If an edge shift expands the segment too much. Concretely, we dilate the original segment with a $7 \times 7$ kernel and check if the new segment is fully contained inside the dilated segment.

**Instance added/removed**: If an edge shift removes a segment or splits a segment into two instances or more.

**Table 2.** Implementation details for learned stages of our pipeline. We use a U-Net implementation from [2], and use the Resnet50 architecture found in the torchvision library. Training time is for the full training set, while testing time is for each floorplan. "Val." stands for validation, where we reserve 20 floorplans from training set and use them to choose best checkpoint.

| | Architecture | # iterations | Batch size | Training time | Testing time | Learning rate | lr. schedule | lr. decay | Val. |
|---|---|---|---|---|---|---|---|---|---|
| Instance segmentation | U-Net | 270K | 16 | 2 days | ∼1 hour | 0.0003 | 6750 | 0.96 | No |
| Type classification | Resnet50 | ∼98K | 64 | 5 hours | <1 minute | 0.001 | ∼9800 | 0.5 | Yes |
| Frame detection | Resnet50 | ∼42K | 32 | 2 hours | <1 minute | 0.001 | ∼4200 | 0.1 | Yes |
| Frame correction | Conv-MPN | ∼420K | 1 | 2 days | ∼1 hour | 0.001 | N/A | N/A | No |

## 5 Implementation details

Our proposed pipeline is implemented in PyTorch. All training and inference can be done on a single NVIDIA V100 GPU with 32G memory. We perform 10-fold cross-validation on our dataset to obtain test-time results for each floorplan. Table 2 shows implementation details for pipeline stages that require learning. In addition, for frame detection training we perform random rotation augmentation at 0, 90, 180, and 270 degrees.

## 6 Evaluations

We compare against one baseline (Mask-RCNN [8]), one state-of-the-art method (Raster-to-Vector [11] denoted as R2V), and a few variants of our system. Our method and Mask-RCNN both perform segmentation, but we predict in smaller local crops and merge predictions. R2V performs corner detection first and then edge/region inference, which our method aims to avoid detecting primitives.
**Mask-RCNN**: We use the detectron2 implemenation [17] using the predefined config of ResNet50+FPN backbone, pretrained on COCO dataset, and fine-tuned for 60k iterations. The learning rate is set to 0.0025. The anchor sizes and ratios are set to (16, 32, 64, 128, 256) and (0.1, 0.5, 1.0, 2.0, 5.0), respectively.
**R2V**: We downloaded and modified the official implementation [11], while considering every boundary as a wall of 2 pixels. All types are considered to be rooms. For example, a door is a line in the R2V system, but a region with a boundary in our problem. The network was trained for 150 epochs with a batch size of 4 on $512 \times 512$ images.

### 6.1 Quantitative results

Besides the standard corner, edge, and region metrics in the existing literature [11,13], we introduce a new one that measures the correctness of the segment "connectivity". In the evaluation, we also consider directions in addition to being just neighbors. Here, we define the connectivity of a segment to be a set of neighboring segments and their directions, where a direction is given by a ray connecting the centroids of the segments.

**Table 3.** Quantitative evaluations are based on the standard corner/edge/region metrics [11] and a new connectivity correctness accuracy. "I", "F", and "H" refers to the the instance segmentation and type classification (Sect. 4.2), the frame detection and correction modules (Sects. 4.3 and 4.4), and the heuristic simplification (Sect. 4.5), respectively. The cyan and the orange color denote the best and the second best results, respectively.

| Model | Corner | | | Edge | | | Region | | | Connectivity Acc. | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Pre. | Rec. | F1 | Pre. | Rec. | F1 | Pre. | Rec. | F1 | All | Frames |
| R2V | 71.3 | 1.9 | 3.7 | 11.5 | 0.3 | 0.5 | 97.8 | 4.5 | 8.7 | 0.0 | 0.0 |
| Mask-RCNN | 6.9 | 25.5 | 10.8 | 0.5 | 1.8 | 0.8 | 44.0 | 15.8 | 23.2 | 0.0 | 35.7 |
| Mask-RCNN + H | 24.6 | 17.1 | 20.2 | 5.1 | 3.2 | 4.0 | 44.0 | 15.8 | 23.2 | 0.0 | 35.7 |
| Ours (I) | 27.2 | 65.4 | 38.4 | 7.6 | 17.2 | 10.6 | 81.7 | 89.3 | 85.3 | 46.6 | 76.7 |
| Ours (I + F) | 27.9 | 63.7 | 38.8 | 8.2 | 17.6 | 11.2 | 81.3 | 89.7 | 85.3 | 48.4 | 80.4 |
| Ours (I + F + H) | 70.4 | 53.1 | 60.5 | 38.3 | 30.4 | 33.9 | 81.2 | 89.7 | 85.2 | 48.4 | 80.3 |

We declare that a segment connectivity is correct if it is exactly equal to the connectivity of the corresponding ground-truth segment with a tolerance of 15 degrees in the directions. Note that we only evaluate the connectivity for segments that are matched with ground-truth segments during region evaluation. Therefore, we report only the accuracy number for the connectivity correctness.

Table 3 shows the main results, comparing against one state-of-the-art (R2V [11]) and one baseline method (Mask-RCNN [8]). For our system, we report numbers at three different stages. Mask-RCNN is a "dense" segmentation method where segment boundaries consist of many corners. Therefore, we also apply our heuristic simplification (Sect. 4.5) after Mask-RCNN. For the connectivity, we report the average accuracy across all segment types as well as only for type "Frame". Note that frames are extremely small segments which pose challenges to existing techniques.

The table shows that R2V and Mask-RCNN both fail with significant margins . Our system after the instance segmentation (I) recovers most regions and achieves high region metrics while suffering severely in corner and edge metrics. The frame detection/correction modules (F) improves the connectivity accuracy for frames by 3.7%. The heuristic simplification (H) further improves the corner and edge metrics significantly.

Table 4 shows the performance of the proposed approach while varying the order and the presence of the three steps. After the instance segmentation (I), it is best to perform frame detection/correction first (F), then the heuristic simplification (H).

### 6.2 Qualitative results

Figures 7 and 8 compare our pipeline against the competing methods. R2V [11] is not designed for varying wall thickness or small segments, and hence fail to recover most segments. Mask-RCNN with heuristic simplification did not


**Fig. 7.** Full-blueprint comparison between our full pipeline and competing methods. "H" stands for heuristic simplification (Sect. 4.5). Our results match the ground-truth data very closely, while other methods cannot recover most instances.

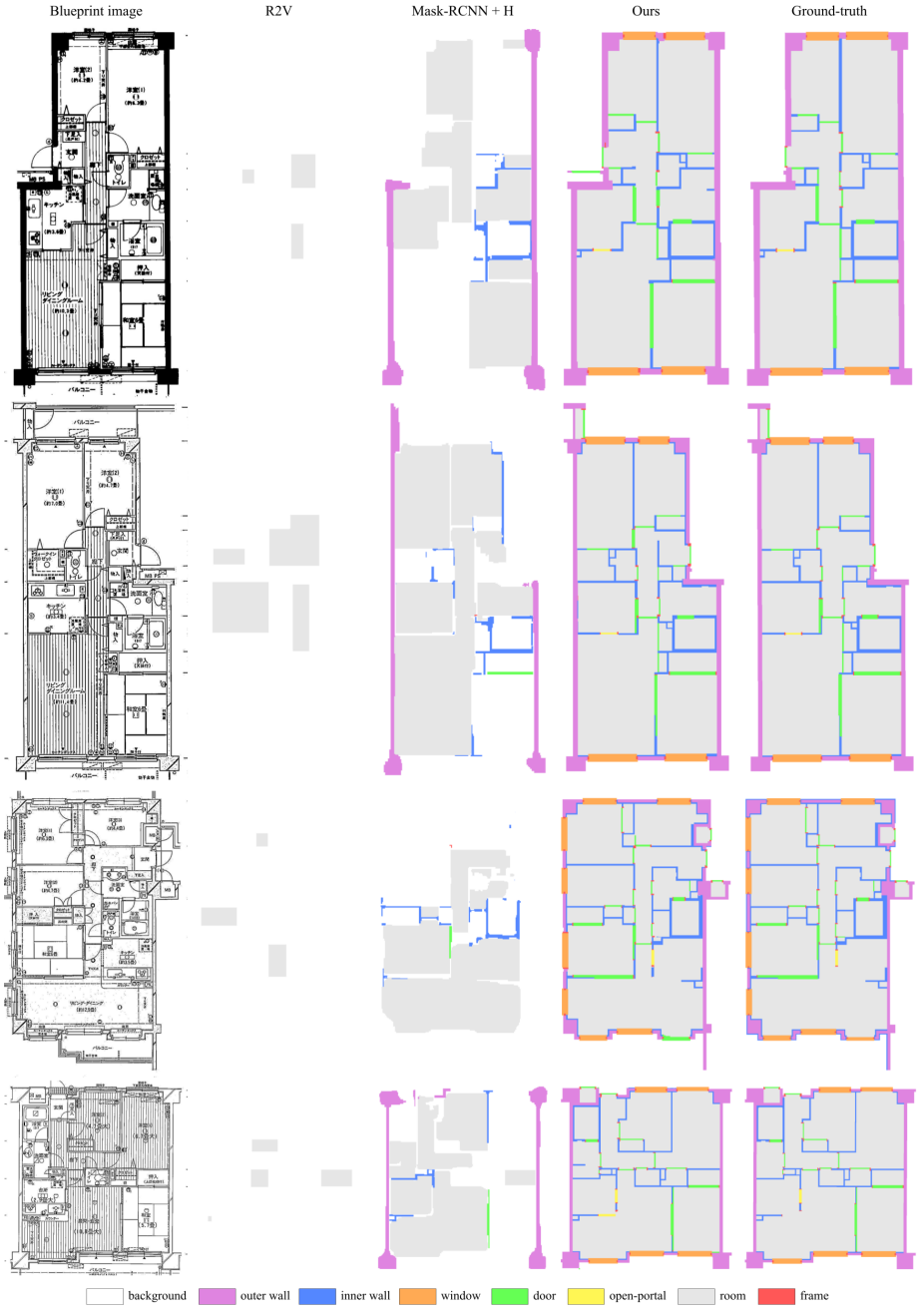


**Fig. 8.** Continued.

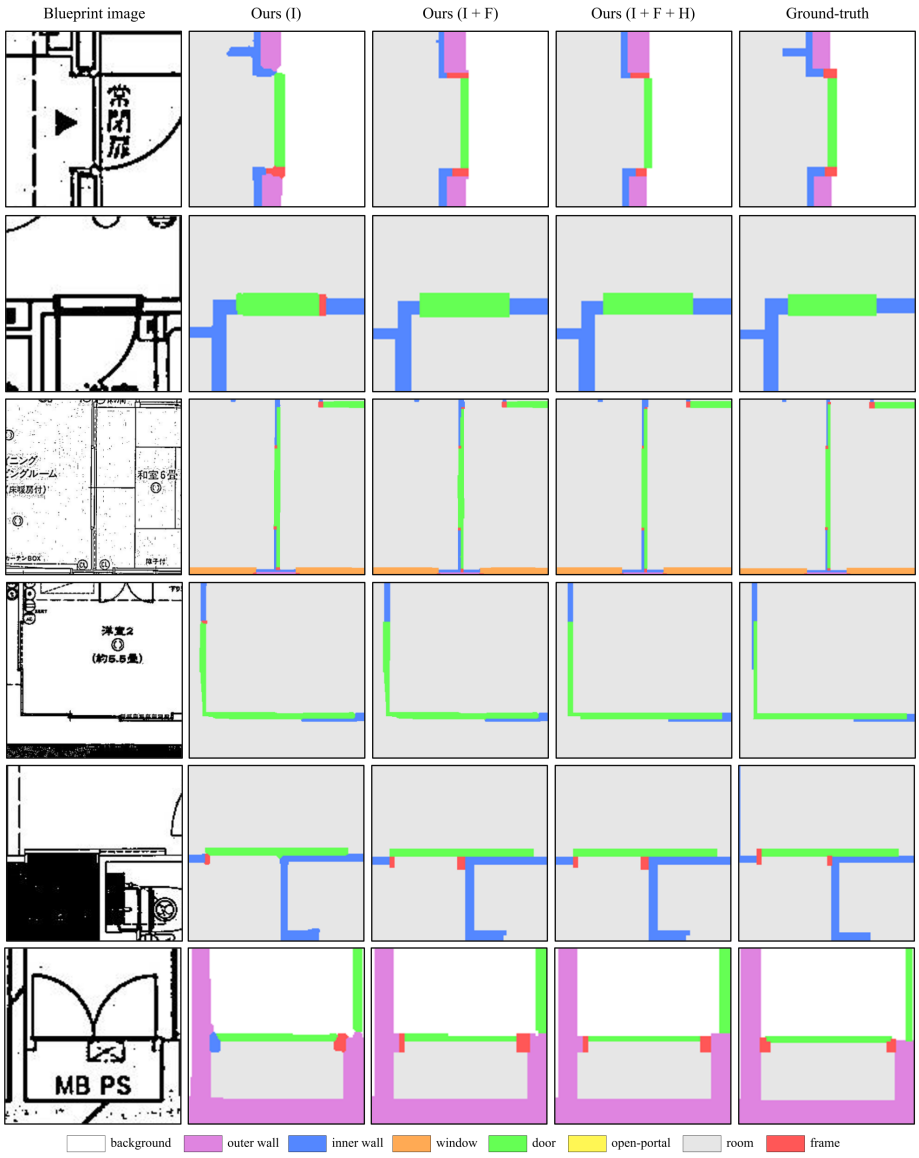


**Fig. 9.** Step-by-step visualization of our pipeline, focusing in around instances. For step names, "I" refers to the instance segmentation and type classification (Sect. 4.2), "F" refers to the frame detection and the correction modules (Sects. 4.3 and 4.4), and "H" refers to the heuristic simplification (Sect. 4.5).

**Table 4.** Ablation study. The table shows the performance of the proposed approach while varying the order and the presence of the three steps, namely, "I" the instance segmentation and the type classification (Sect. 4.2), "F" the frame detection and the correction modules (Sects. 4.3 and 4.4), and "H" the heuristic simplification (Sect. 4.5). The cyan and the orange color denote the best and the second best results, respectively.

| Config | Corner | Edge | Region | Connectivity | Acc. |
|---|---|---|---|---|---|
| | F1 | F1 | F1 | All | Frames |
| Ours (I) | 38.4 | 10.6 | 85.3 | 46.6 | 76.7 |
| Ours (I + H) | 61.1 | 34.1 | 85.3 | 46.6 | 76.7 |
| Ours (I + F) | 38.8 | 11.2 | 85.3 | 48.4 | 80.4 |
| Ours (I + H + F) | 59.4 | 32.5 | 85.2 | 48.1 | 79.8 |
| Ours (I + F + H) | 60.5 | 33.9 | 85.2 | 48.4 | 80.3 |

perform well, producing results that are far from complete. Our method clearly outperforms the two.

Figures 9 evaluate the quality of our frame connectivity. A mix of adding or removing actions and different frame connections are shown, where our system successfully corrects frame connectivity. Refinement GAN based simplification removes many jagged edges, followed by heuristic simplification which produces further visually-pleasing results.

The second row of Figure 9 is a common case for bathroom doors, where there should be no door frames. Our pipeline removed the extra frame and filled in the open region. The first row is a common example of doors with two frames. Here, we see a failure case of the frame correction module where the top left horizontal blue wall disappears due to the refinement procedure. The last row is interesting where a door frame was segmented, but mistakenly classified as a blue wall. Our system is able to handle this mistake, and also smooth the green door's jagged line with our heuristic simplification.

## 7 Conclusion

This paper proposes a novel image vectorization algorithm for building blueprints, whose geometric structures are far more complex and detailed than standard floorplan images. Qualitative and quantitative evaluations demonstrate the effectiveness of our approach, outperforming the current state-of-the-art and a baseline method with significant margins. However, the precision and recall are still far from the production quality, where our main future work will be to further robustify the algorithm.

**Acknowledgements** The research is supported by NSERC Discovery Grants, NSERC Discovery Grants Accelerator Supplements, and DND/NSERC Discovery Grants. We also thank GA Technologies for providing us with the building blue-print images.

## References


1. instance-seg. https://github.com/alicranck/instance-seg, accessed: 2021-04-16 6
2. Unet: semantic segmentation with pytorch. https://github.com/milesial/Pytorch-UNet, accessed: 2021-04-16 9
3. Ahmed, S., Liwicki, M., Weber, M., Dengel, A.: Improved automatic analysis of architectural floor plans. In: 2011 International conference on document analysis and recognition. pp. 864–869. IEEE (2011) 2
4. Cao, Z., Hidalgo, G., Simon, T., Wei, S.E., Sheikh, Y.: Openpose: realtime multi-person 2d pose estimation using part affinity fields. IEEE transactions on pattern analysis and machine intelligence **43**(1), 172–186 (2019) 3
5. Chen, J., Liu, C., Wu, J., Furukawa, Y.: Floor-sp: Inverse cad for floorplans by sequential room-wise shortest path. In: The IEEE International Conference on Computer Vision (ICCV) (2019) 4
6. De Brabandere, B., Neven, D., Van Gool, L.: Semantic Instance Segmentation with a Discriminative Loss Function. In: CVPR Workshop on "Deep Learning for Robotic Vision" (2017) 3, 5, 6
7. Dutta, A., Zisserman, A.: The VIA annotation software for images, audio and video. In: Proceedings of the 27th ACM International Conference on Multimedia. MM '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3343031.3350535, https://doi.org/10.1145/3343031.3350535 4
8. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: 2017 IEEE International Conference on Computer Vision (ICCV). pp. 2980–2988 (2017). https://doi.org/10.1109/ICCV.2017.322 3, 5, 9, 10
9. de las Heras, L.P., Terrades, O.R., Robles, S., Sánchez, G.: Cvc-fp and sgt: a new database for structural floor plan analysis and its groundtruthing tool. International Journal on Document Analysis and Recognition (IJDAR) **18**(1), 15–30 (2015) 2
10. Kalervo, A., Ylioinas, J., Häikiö, M., Karhu, A., Kannala, J.: Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis. In: Scandinavian Conference on Image Analysis. pp. 28–40. Springer (2019) 5
11. Liu, C., Wu, J., Kohli, P., Furukawa, Y.: Raster-to-vector: Revisiting floorplan transformation. In: 2017 IEEE International Conference on Computer Vision (ICCV). pp. 2214–2222 (2017). https://doi.org/10.1109/ICCV.2017.241 2, 3, 4, 9, 10
12. Liu, C., Wu, J., Furukawa, Y.: Floornet: A unified framework for floorplan reconstruction from 3d scans. In: Proceedings of the European Conference on Computer Vision (ECCV) (September 2018) 4
13. Nauata, N., Furukawa, Y.: Vectorizing world buildings: Planar graph reconstruction by primitive detection and relationship inference. In: European Conference on Computer Vision. pp. 711–726. Springer (2020) 4, 9
14. Nauata, N., Hosseini, S., Chang, K.H., Chu, H., Cheng, C.Y., Furukawa, Y.: House-gan++: Generative adversarial layout refinement networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2021) 7
15. Newell, A., Yang, K., Deng, J.: Stacked hourglass networks for human pose estimation. In: European conference on computer vision. pp. 483–499. Springer (2016) 3
16. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems. vol. 28 (2015) 3

17. Wu, Y., Kirillov, A., Massa, F., Lo, W.Y., Girshick, R.: Detectron2. https://github.com/facebookresearch/detectron2 (2019) 9
18. Zeng, Z., Li, X., Yu, Y.K., Fu, C.W.: Deep floor plan recognition using a multi-task network with room-boundary-guided attention. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019) 5
19. Zhang, F., Nauata, N., Furukawa, Y.: Conv-mpn: Convolutional message passing neural network for structured outdoor architecture reconstruction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020) 4, 7