From Floor Plans to Virtual Reality

Timothée Fréville, Charles Hamesse, Benoit Pairet, Rihab Lahouli, Rob Haelterman
XR Lab - Department of Mathematics
Royal Military Academy
Brussels, Belgium
timothee.freville@mil.be charles.hamesse@mil.be

***Abstract*—Creating realistic VR environments is a tedious task. For many types of applications, these environments must respect certain constraints or correspond as faithfully as possible to a real place. Floor plans are a simple and abundant format that humans can read and edit. As such, they are a good basis to create VR environments that match real buildings. We propose a method to convert floor plans to VR environments with minimal human intervention. Leveraging traditional computer vision, machine learning and 3D engines, our method is efficient but remains flexible and fast, and creates simple yet realistic environments that can be used for various VR applications. We demonstrate results for our specific use case for Belgian Defence's tactical intervention teams.**

***Index Terms*—Floor plans, Virtual Reality, 3D environments**

I. Introduction

Generating VR environments is a notoriously tedious task. Depending on the target application, different levels of attention must be brought to different aspects of the environment. Aiming for maximum photorealism is not always required, for example in the case of serious gaming or procedure training applications where the place, scale, behaviour and lifecycle of objects are most important. For example, in the case of firefighting training, rooms must feature doors, windows and flammable props at relevant locations chosen by the instructor.

Floor plans are an easily understandable and editable format. Moreover, they can be sketched quickly with widely available software or even by hand. With the advent of machine learning for computer vision, methods for automatically parsing floor plans have appeared. This allows software programs to automatically create a digital representation of the floor plan, which can then be turned into a 3D model. Then, this 3D model can be turned into a VR environment. This is the work we describe in this paper: a framework to turn floor plans into VR environments automatically.

In addition to presenting our framework, we demonstrate its usage for a solution much sought after by Belgian Defence's tactical intervention teams: being able to train in new environments that fit a rough description of a building: room geometry, large pieces of furniture and most importantly: location of doors and windows to enter and exit the premises.

II. Related work

Traditional methods to parse floor plans use classical image processing methods to specific features which might indicate the presence of walls, doors or similar objects. [8] uses image vectorization and a Hough transform to perform line detection. If the lines satisfy a specific graphical arrangement, they are combined into walls. A similar method is proposed to extract arcs and detect door hypotheses. An alternative method that uses patch-based segmentation with visual words and does not need image vectorization is proposed in [5]. In [2], an algorithm is designed to differentiate between thick, medium, and thin lines to detect walls and remove the components outside the convex hull of the outer walls. In [4], noise removal techniques, erosions and dilations are used to extract a thick and a thin representation of the walls. The thin one is then subtracted from the thick one, which results in a hollow representation of walls with a constant thickness that can then be used as a reference for 3D modeling.

More recently, deep learning-based methods have been introduced. For example, [14] proposes a multi-task neural network to learn to predict room-boundary elements (walls, windows, doors) and room types (bedroom, kitchen, etc). This is done using a shared encoder to extract features from the floor plan image and one separate decoder for each task. The architecture of the encoder and the decoders are based on VGG [12]. In [15], an object detector based on Faster-RCNN [11] is fine-tuned to learn to predict annotations in various floor plan datasets. A similar approach is proposed in [13], but this time using a variant of the YOLO object detector [10]. The floor plan datasets used in this work include [9] and [3].

The field of 3D environment generation has progressed dramatically in the last few years as it is heavily pushed by the gaming industry. 3D engines such as Unity [7] or Unreal Engine [6] are becoming more and more straightforward to use, flexible and powerful. Environments can be assembled from 3D props designed (and if need be, animated) beforehand, manually or programmatically. A vast quantity of 3D models is available on the marketplace, which makes the development of new environments easier.

In this work, we combine traditional computer vision techniques to detect room boundary features (walls, doors and windows) with a deep learning-based method to detect the interior objects and furniture (sofa, bed, etc) and ad-hoc map generation scripts for Unreal Engine to turn floor plans into VR-ready environments.

The research presented in this paper has been funded by the Belgian Royal Higher Institute for Defense, in the framework of the project DAP18/04.

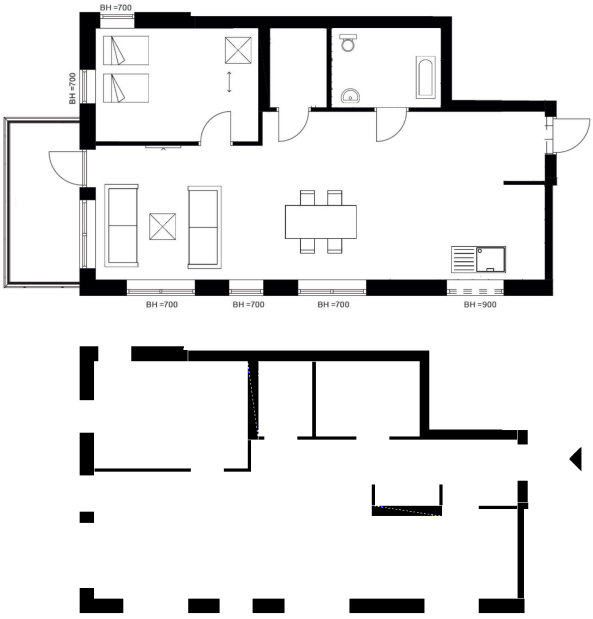

Fig. 1. Wall detection. Top: original floor plan. Bottom: extracted walls. The image was a digital copy without any noise, taken from [4].

III. Method

We describe the different methods that we use to detect walls, windows, doors and interior objects.

A. Detecting walls

Our method is based on [4], which mainly uses traditional computer vision methods and is implemented using OpenCV. The steps of the procedure to go from a grayscale floor plan image to a list of wall coordinates are the following:

1) Conversion to a binary image using binary inverse thresholding.
2) Noise removal method with opening (erosion then dilation) to remove the thin details (e.g. furniture). At this point, we have an image representing a thick version of the walls.
3) Distance transform: the value of each pixel belonging to a wall is replaced by the distance between this pixel and the nearest black pixel (i.e. the nearest edge of the wall)
4) Conversion of this distance image to a binary image, again with binary thresholding. This results in an image representing a thin version of the walls.
5) Subtraction of thin walls (Step 4) from thick walls (Step 2), which results in a representation of hollow walls with a constant thickness.

A sample execution of this procedure is shown in Figure 1.

B. Detecting windows and doors

Detecting the location of windows and doors is done using template matching and a specific procedure to distinguish between doors and windows. Again, we use OpenCV for this part. First, we use a sample door as the reference patch from which we extract ORB features. Then, we search for similar features in the floor plan with a brute-force matcher. However, this matching method returns multiple occurrences per door in the floor plan. Therefore, we develop an algorithm to refine these door proposals by keeping only the ones where the feature points are found on the hole of a wall between two rooms. Windows are then put at every hole in the walls that are on the outer contour of the floor plan, i.e. they do not serve as junctions between two rooms. Results of this procedure are shown in Figure 2.

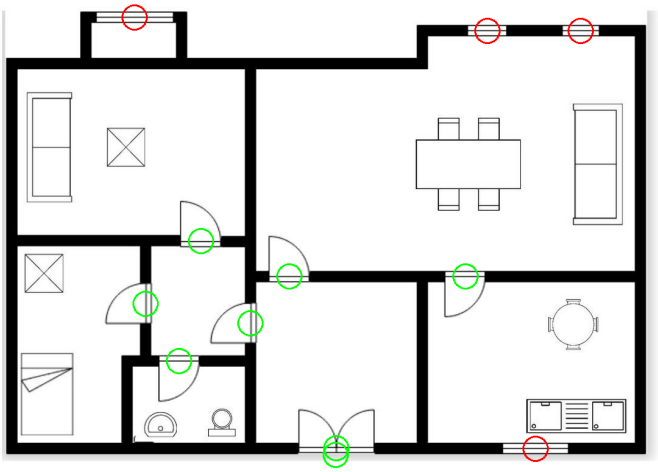

Fig. 2. Detection of windows (red) and doors (green).

C. Detecting interior objects

Our implementation is based on [13], i.e. a YOLO object detector fine-tuned on floor plan datasets with 12 classes (sofa, sink, bed, etc). It is implemented using TensorFlow [1]. In addition to using the YOLO object detector to parse the content of the room, we implement a post-processing method to avoid duplicate detections: for each pair of overlapping

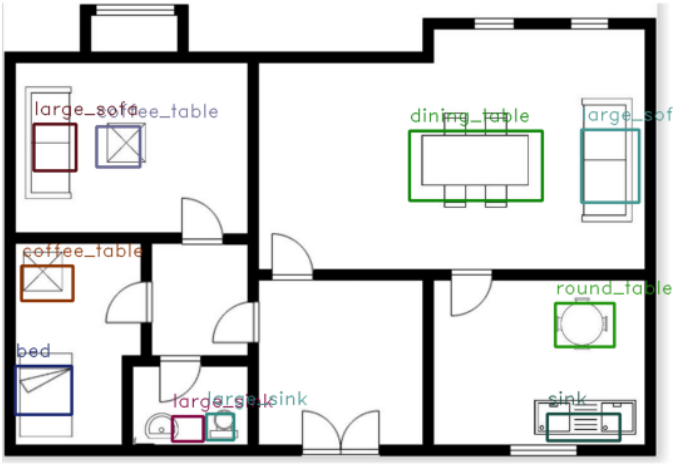

Fig. 3. Detection of interior objects.

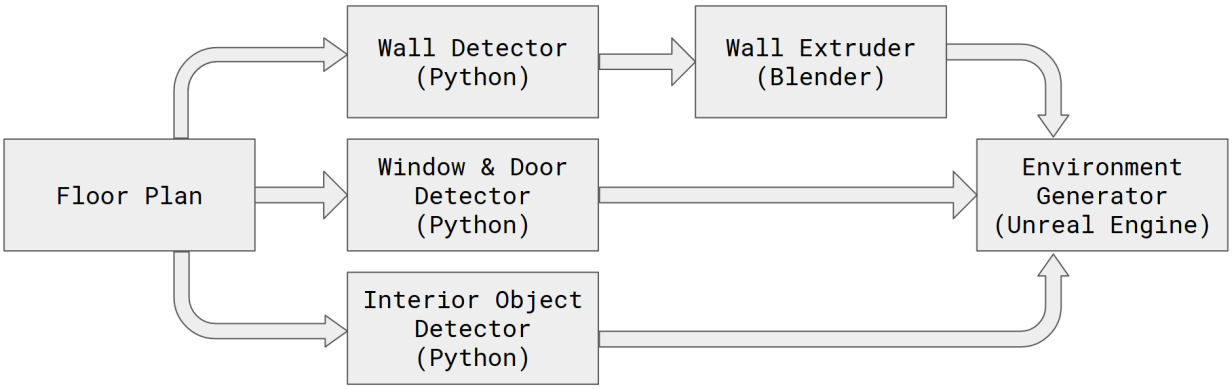

Fig. 4. Overview of our system.

bounding boxes, we keep only the one with the maximum confidence. Example detections are shown in Figure 3.

D. Generating the VR environment

We use Unreal Engine and the Python API to create the environment in an automated manner. We implement a script that takes the list of objects returned from our previous software components and spawns the related assets in the 3D world at their respective position. To do so, we maintain a list of the assets per type: sink, bed, sofa, double sofa, etc. In addition to the walls, doors and windows, our method can spawn 12 different types of objects.

E. Introducing randomness

To create environments that match the constraints imposed by the floor plans but still exhibit some variety, we implement a method that selects a given 3D asset randomly in a pre-defined list for each asset type (sofa, dining table, etc).

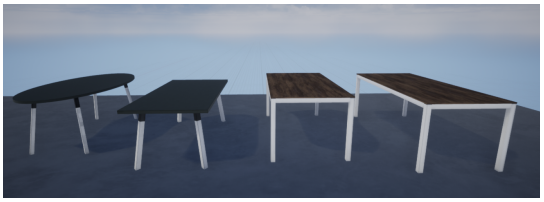

Fig. 5. Various dining table assets used in our system.

Our entire framework is depicted in Figure 4.

IV. Evaluation

Our evaluation consists in taking three floor plans relevant for our use case, running our method and visually assessing the quality of the resulting environments. The floor plans in this section were picked from internet searches and [4]. Our tests are performed using a computer with an Intel i7 8-core CPU and an Nvidia RTX2080Ti GPU. In terms of computational time, the execution of our method only takes a few seconds: running the detection algorithm (including the neural network inference) takes but a fraction of a second, then a few seconds are needed to instantiate the Unreal Engine level with the necessary assets.

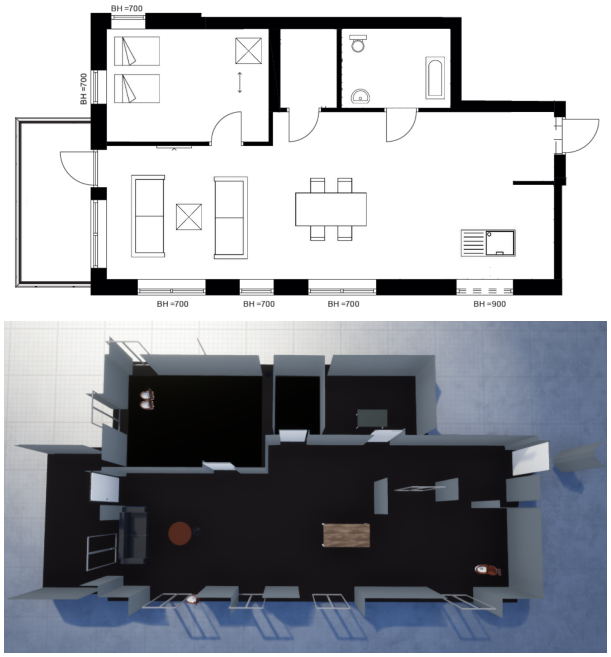

Fig. 6. Experiment on the the first test floor plan. Top: original floor plan. Bottom: 3D environment.

Fig. 7. Outcomes from the 2nd plan with the VR vision

Results are shown in Figure 6 to Figure 11. While our method allows to generate environments that match the floor

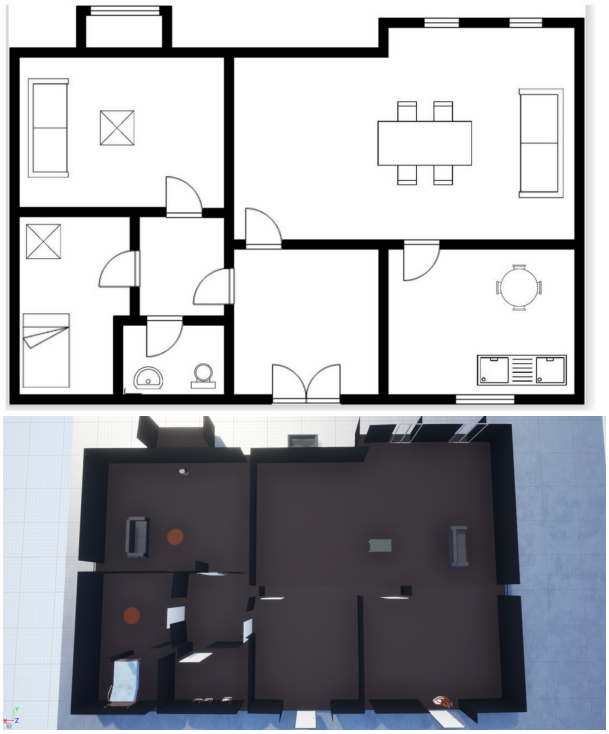

Fig. 8. Outcomes from the 2nd plan

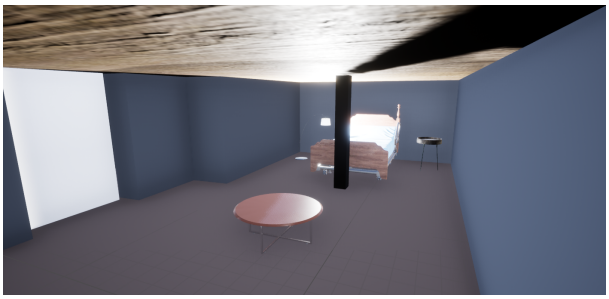

Fig. 9. Outcomes from the 2nd plan with the VR vision

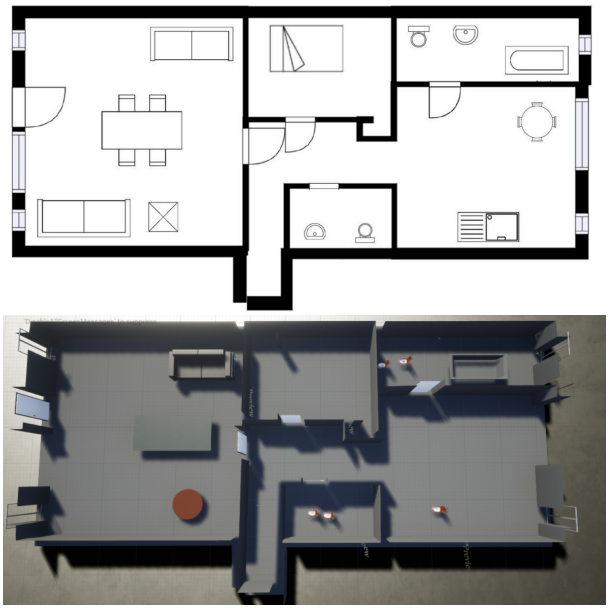

Fig. 10. Outcomes from the 3rd plan

Fig. 11. Outcomes from the 3rd plan with the VR vision

geometry with minimal human intervention, in some cases details can be missing or slightly wrong. For example, the size of the windows and doors could be adjusted to better match the holes in the walls. Using textures for the walls of rooms would increase photorealism. Moreover, there a few occurrences of misclassified doors and windows, which indicates that there is room for improvement on the detectors too.

V. Conclusion

We have developed an automated framework to turn floor plans into VR environments that feature walls, doors, windows, and large pieces of furniture. Our system allows to build VR-ready environments that depict the geometry of entire floors with the locations of doors and windows, critical for tactical intervention teams. As experienced during the tests, there is room for improvement in each of the components of the framework. Our approach is modular, so it will be convenient to achieve this further work on each component independently.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Sheraz Ahmed, Marcus Liwicki, Markus Weber, and Andreas Dengel. Improved automatic analysis of architectural floor plans. In *2011 International Conference on Document Analysis and Recognition*, pages 864–869, 2011.

[3] Chiranjoy Chattopadhyay . Repository of building plans (robin). 2019.

[4] Daniel Westberg. Floor plans to blender 3d. 2019.

[5] Lluis-Pere de las Heras, Joan Mas, Gemma Sánchez, and Ernest Valveny. Wall patch-based segmentation in architectural floorplans. In *2011 International Conference on Document Analysis and Recognition*, pages 1270–1274, 2011.

[6] Epic Games. Unreal engine. 2019.

[7] John K Haas. A history of the unity game engine. *Worcester Polytechnic Institute*, 2014.

[8] Sébastien Macé, Hervé Locteau, Ernest Valveny, and Salvatore Tabbone. A system to detect rooms in architectural floor plan images. pages 167–174, 06 2010.

[9] Mathieu Delalandre. Systems evaluation synthetic documents (sesyd). 2019.

[10] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. pages 6517–6525, 07 2017.

[11] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 91–99, Cambridge, MA, USA, 2015. MIT Press.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[13] Deewan Singh. Object detection in floor plan images. 2019.

[14] Z. Zeng, X. Li, Y. Yu, and C. Fu. Deep floor plan recognition using a multi-task network with room-boundary-guided attention. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9095–9103, Los Alamitos, CA, USA, nov 2019. IEEE Computer Society.

[15] Zahra Ziran and Simone Marinai. Object detection in floor plan images. In Luca Pancioni, Friedhelm Schwenker, and Edmondo Trentin, editors, *Artificial Neural Networks in Pattern Recognition*, pages 383–394, Cham, 2018. Springer International Publishing.