Object Detection in Floor Plans for Automated VR Environment Generation

Timothée Fréville, Charles Hamesse, Bênoit Pairet and Rob Haelterman
Royal Military Academy, Rue Hobbema 8, Brussels, Belgium

Keywords: Image Recognition, Floor Plans, Neural Networks, Synthetic Data.

Abstract: The development of visually compelling Virtual Reality (VR) environments for serious games is a complex task. Most environments are designed using game engines such as Unity or Unreal Engine and require hours if not days of work. However, most important information of indoor environments can be represented by floor plans. Those have been used in architecture for centuries as a fast and reliable way of depicting building configurations. Therefore, the idea of easing the creation of VR ready environments using floor plans is of great interest. In this paper we propose an automated framework to detect and classify objects in floor plans using a neural network trained with a custom floor plan dataset generator. We evaluate our system on three floor plans datasets: ROBIN (labelled), PFG (our own Procedural Floor plan Generation method) and 100 labelled samples from the CubiCasa Dataset

1 INTRODUCTION

Designing high-quality VR environments is a notoriously tedious but important task for the succesful deployment of VR applications. 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 the best option, for example in the case of serious gaming or procedure training applications where the place, scale, behaviour and lifecycle of objects are the 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 (Haelterman et al., 2020).

Floor plans are an easily understandable and editable format. This representation also has the advantage to be universal. Furthermore, with the democratization of the use of software to design these plans, architects and individuals are now able to create and modify floor plans in a few minutes. Adding a VR ready experience to this system is a natural prolongation of the technology. Furthermore, displaying interior layout using a VR headset can have much more impact than a regular computer screen (Portman et al., 2015) and (Davila Delgado et al., 2020).

In this work, we extend the framework proposed in (Fréville et al., 2021). More specifically, we present enhancements done on the object detection component of the framework. In this case, the most important aspect is the presence and placement of objects that will influence the progression of tactical squad: doors, windows, room and building layouts. Our contributions are the following:

- A floor plan dataset generator with various image deterioration methods (noise, distortion, etc) and ideal ground truth, to train a neural network with robustness;
- A neural network trained on this dataset for a more versatile detection and a thorough evaluation on real-life floor plans;
- The integration of our method in a more generic framework for converting floor plans into VR environments.

2 RELATED WORK

We review the state-of-the-art methods in floor plan parsing, 3D environment generation for VR and complete systems to generate VR environments from floor plans.

2.1 Floor Plan Parsing

2.1.1 Classical Approaches

Traditional floor plan parsing techniques use conventional image processing techniques to target ele-

Fréville, T., Hamesse, C., Pairet, B. and Haelterman, R.
Object Detection in Floor Plans for Automated VR Environment Generation.
DOI: 10.5220/0011629300003417
In *Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2023) - Volume 5: VISAPP*, pages 480-486
ISBN: 978-989-758-634-7; ISSN: 2184-4321

ments that could point to the presence of walls, doors, or other such things. (Macé et al., 2010) performs line detection using image vectorization and a Hough transform. If a particular graphical arrangement is met, the lines are merged to form walls. The authors suggest a similar technique to extract arcs and find door hypotheses. (de las Heras et al., 2011) proposes an alternative approach that does not require image vectorization and employs patch-based segmentation with visual words. The method in (Ahmed et al., 2011) is built to distinguish between thick, medium, and thin lines in order to identify walls and eliminate any components that are outside the convex hull of the outer walls. In (Daniel Westberg, 2019), a thick and a thin representation of the walls are extracted using noise removal techniques, erosions, and dilations. Then, a hollow representation of walls with a constant thickness is created by subtracting the thin representation from the thick one, which may then be used as a reference for 3D modeling.

2.1.2 Deep Learning-Based Methods

To learn to predict room-boundary elements (e.g. walls, windows, and doors) (Zeng et al., 2019) describes a multi-task neural network. In order to extract features from the floor plan image, a shared encoder is used, and one decoder is used for each task. Both the encoder and the decoders' architectures are based on VGG (Simonyan and Zisserman, 2015). A Faster-RCNN-based (Ren et al., 2015) object detector is enhanced in (Ziran and Marinai, 2018) to learn to anticipate annotations in diverse floor plan datasets. The same strategy is suggested in(Singh, 2019), however this time it uses of a modified version of the YOLO object detector (Redmon and Farhadi, 2017). Training or evaluation of these neural networks is done on the datasets (Mathieu Delalandre, 2019) and (Chiranjoy Chattopadhyay , 2019), which we also use in this study.

2.2 3D Environment Generation

The game industry has been a major driver of advancement in the field of 3D environment generation during the past few years. 3D engines like Unity (Haas, 2014) and Unreal Engine (Epic Games, 2019a) are becoming easier to use, more adaptable, and more potent. Environments can be created using 3D props that have been manually or programmatically created (and, if necessary, animated). The market has a huge selection of 3D models, which facilitates the creation of new surroundings. Blender (Foundation, 2002) is also a good choice. It is open-source and has an active community and plugins to execute tasks programmatically from Python scripts. Since Blender does not have as many interaction features or VR capabilities like game engines, we opt to develop our algorithm with game engines. More particularly we choose Unreal Engine 4 because of its Virtual Reality integration plugin that works perfectly with Steam VR (Valve, 2003), ensuring the versatility of our model on various VR platforms such as Oculus (Meta, 2012) or Varjo (Varjo, 2016). Furthermore, the access to the Unreal Market (Epic Games, 2019b) is an important feature that allows us to ease the process of our implementation by getting plugins and assets from the community.

2.3 Floor Plan to VR Environment Frameworks

There exists many frameworks that achieve the tasks of floor plan parsing and 3D environment generation. 3DPlanNet uses heuristic rules to retrieves walls and Tensorflow Object detection API for windows and doors (Park and Kim, 2021). This study (Dodge et al., 2017) uses a fully connected neural network and uses OCR (Optical Character Recognition) to estimate the size of the rooms. The framework proposed in (Fréville et al., 2021) combines traditional computer vision and deep-learning techniques to detect room boundary features (walls, doors and windows, interior objects). They implement ad-hoc map generation scripts for Unreal Engine to turn floor plans into VR-ready environments. In our case, we use a neural network-based solution for object detection using a Yolo instance (Redmon and Farhadi,). To ensure sufficient training data, we design a floor plan generator with random image perturbations. Doing so allows us to feed YoloV5 with virtually unlimited training data, allowing our implementation to be more robust.

3 METHOD

In this section, we present our floor plan parser and our custom dataset generator. The architecture of the parser system, depicted in Figure 3 has two main two parts :

1. The furniture recognition : which uses the YOLOv5 Framework (Deep Learning methods)
2. The wall recognition that uses OpenCV libraries (Traditional image processing tool)

3.1 Walls Detection

Our method is based on (Daniel Westberg, 2019), 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: Each pixel that is a part of a wall has its value changed to reflect the separation between it and the closest black pixel, or the wall's edge.
4. This distance image is converted to a binary image once again using binary thresholding. As a result, the walls appear narrow in the resulting image.
5. A representation of hollow walls with a constant thickness is produced by subtracting thin walls (Step 4) from thick walls (Step 2) in the process.

3.2 Object Detection

The most recent edition of YOLO is the v5 (Redmon and Farhadi,). The backbone is composed of a Cross-stage partial networks CSP (Wang et al., 2019) are employed in YOLOv5 to extract valuable features. For feature pyramid retrieval, YOLOv5 leverages PANet (Liu et al., 2018). Then, YOLOv5 uses the sigmoid activation function in the final detection layer and the leaky ReLU activation function in the middle/hidden layers. For naturally occurring photos, YoloV5 is one of the most rapid and precise object detection method.

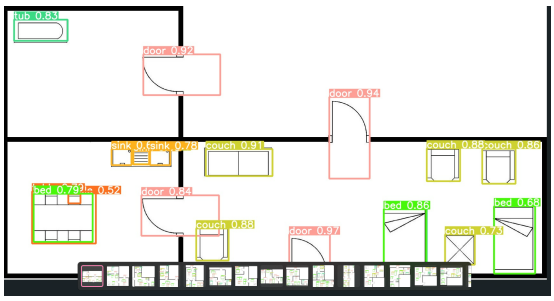

Figure 1: YoloV5 on inference.

YoloV5 has shown its effectiveness to recognize 2D features on hand-drawn images (Reddy and Panicker, 2021). Therefore this neural network framework has been chosen for the furniture recognition part. The training was done using a multi-GPU setup using GTX Tesla cards, a batch size of 16, learning rate of 0.001, a momentum of 0.937 and 500 epochs. Using YOLOv5 allows us to benefit from the extensive inference and test metrics implemented by default in the framework.

3.3 3D Environment Generation

The 3D generation is achieved using Unreal Engine 4. Therefore we used a plugin found on the marketplace to import the meshes in runtime (Virtual Bird, 2018). Furthermore we create our own plugin to transform YOLOv5 text file output into readable object by Unreal Engine.

3.4 Training Data Generator

To train a deep neural network we need numerous labeled images. Nowadays many datasets are available online for common object detection (e.g: airplanes, cars, boat, animals, etc) on various websites such as Kaggle, Robotflow or Github. This considerably eases the neural network learning process as the quality of the neural network highly depends on the data feed. Unfortunately, specific object detection requires specific datasets. Therefore, a vast quantity of labeled floor plans are required for our application. This leads to two issues. First, this sort of dataset is relatively hard to find. Second, floor plan data found in one source are most likely to use the same symbols for furniture which increases the probability of over-fitting and does not help our model to be robust during inference. Our solution was to program a floor plan generator that can create thousands of labelled data using different layouts.

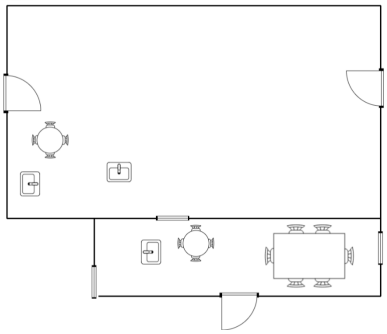

Figure 2: Example of floor plan generated.

Creating an accurate and realistic floor plan image by using a generator can quickly become a tedious task if we do not restrain the scope on the most important aspect: interior object recognition. The aim here is to merge the window and door recognition with the

furniture recognition. Therefore, creating an image with a convincing room configurations and usable furniture arrangement does not enter into our consideration. The neural network only needs to learn object shapes and basic placement rules (e.g. doors and windows are mostly linked with walls). Using the known rules and random functions we can develop a program to generate thousands of labeled floor plans with different floor layouts and symbols.

3.5 Complete System

Figure 3 represent the final architecture of our method. Using multiple python scripts in parallel and Docker we can easily generate a 3D representation of the world using Unreal Engine 4 as our main virtual engine.

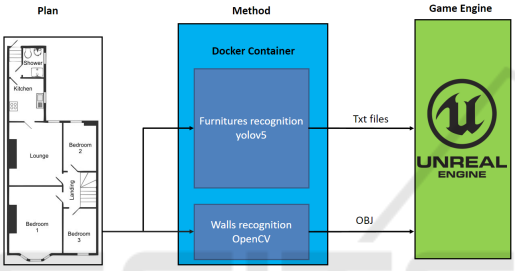

Figure 3: Final Architecture.

4 EVALUATION

The military end-users would have different level of detail of floorplans in their possession.

- Some of them would be accessible directly from the architect plan achieved using a software. Which means almost zeros noise and high resolution plans. We are testing this using a synthetic dataset named ROBIN.
- The second option would be old plans scanned, which obviously features more noise and defaults induced by the age of the paper itself or the drawing done by the architect. We would be testing it using Cubicasa datasets which feature scans of real-world floor plans.
- The final option would be to draw by hand directly on paper the floorplan if no other option is available. This features becomes quite interesting, allowing the end-user to draw virtually hundred of plans for tactical training. For this we will test on drawn floor plan made by our self.

We evaluate the performance of our methods on a machine with an Intel i7 8-core CPU and an Nvidia RTX 2080Ti GPU. The computational time required to carry out our method is extremely brief: only a few seconds are needed to run the detection procedure, which includes the neural network inference. The use of Docker allows us to use pre-configured environments with libraries and tools to run codes. Using this technique we greatly enhance the number of VR environments generated. The YOLO instance is launched to detect every image present on a folder. Since the implementation is splitting the generation of walls and the furniture detection, it is much more efficient and reliable to only evaluate our model on the furniture part. Firstly, it is a tedious task to find a floor plan dataset that has walls labelled. Therefore the inference on furniture would be our only metrics. Our model only involves the position and the size of the objects found, not rotation. However, in our use case the rotation can be easily guessed as objects such as sinks, couches or toilets are in most apartments turned towards the center of the room. Furthermore, objects as tables or plants do not need rotation information to be accurate. The evaluation part has been split in two parts: first our detection system is evaluated with the ROBIN synthetic dataset. In a second time it will be evaluated with a real-world dataset. Those datasets will be scans of buildings available from real estate companies. Three different metrics have been used for the evaluation part:

mAP: Mean Average Precision : The mAP calculates a score by comparing the detected box to the ground-truth bounding box. The model's detections are more precise the higher the score.

Precision: Precision is a model's capacity to recognize only the pertinent objects. A model with a precision of 1.0 produces no false positives. However, even if there are bounding boxes that should be identified but aren't, the value will still be 1.0.

$$Precision = TP/(TP+FP) \quad (1)$$

Recall: Recall is a model's capacity to locate every ground truth bounding box. A model with a recall of 1.0 creates no false negatives, or undetected bounding boxes that should be identified. The recall will still be 1.0, even if there is a "overdetection" and the incorrect bounding box is discovered.

$$Recall = TP/(TP+FN) \quad (2)$$

4.1 Synthetic Data (ROBIN)

Many research papers use the ROBIN dataset to validate their framework: (Sharma et al., 2018) uses

it to test the performance of their neural network to match floorplans with hand-drawn sketch plans. (Goyal et al., 2019) uses the dataset to generate more floor plans. Better results are expected with this type of data since it is noise- and distortion-free. The labelling of this dataset has been done using Robotflow and can be accessed in (Timothée Fréville, 2022) for further investigation. Therefore, it is a perfect case to test our algorithms on a synthetic dataset that was not in the training set. If the inference test is satisfactory we could switch to a real world dataset.

Table 1: Results on ROBIN datasets.

Object	mAP	Precision	Recall
Doors	0.774	0.713	0.771
Table	0.361	0.695	0.082
Sink	0.254	0.546	0.327
Couch	0.496	0.643	0.259
Bed	0.280	0.317	0.699
Toilet	0.738	0.684	0.759
Tub	0.743	0.650	0.836
All	0.536	0.606	0.533

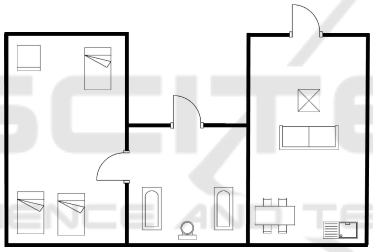

Figure 4: Sample of the ROBIN dataset.

The ROBIN dataset does not feature window objects. As we can see the model confuses objects. This can be explained by the symbol system used by the ROBIN dataset that differs significantly from the ones present in the dataset generated to train YOLOv5 (e.g. tables are represented by a square with a cross which, model has never been trained with such representation). Furthermore, sinks and beds differ a lot compared to the training dataset used Fig5. However, the model detects almost every object on plans but the recognition can definitely be enhanced.

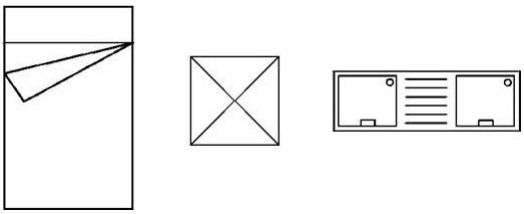

Figure 5: Objects that are hardly recognized by our framework.

4.2 Real-World Data (CubiCasa)

The CubiCasa dataset does not feature objects of couch, bed and toilet classes. In particular, the CubiCasa5K version is a large-scale floor plan image collection with 5000 samples with objects that have been classified into more than 80 categories. By employing polygons to divide the various objects, the dataset annotations are more precise than the usual ones using bounding boxes. This dataset suits our need of diverse symbols and floor plan layouts in general. Most of the images feature distortion and noise as most of the images are pictures or scans taken from the real plans.

Table 2: Results on CubiCasa datasets.

Object	mAP	Precision	Recall
Window	0.353	0.585	0.162
Doors	0.461	0.514	0.408
Sink	0.107	0.285	0.027
Toilet	0.074	0.153	0.016
Tub	0.016	0.047	0.021
All	0.202	0.316	0.126

Figure 6: Sample of the Cubicasa dataset.

As expected, the model gets more frequently lost against a real-world dataset. Most of the objects are confused with the background. The real-world dataset features pen-drawing, distortion, colors for certain plan and an overlapping of symbol that confuses the recognition.

4.3 Hand-Drawn Dataset

This hand-drawn dataset has been created specially for this study. This is a feature specially interesting for the military forces, as they can use it draw the floor worthy of interest and quickly train for intervention.

This dataset is composed of seven plans that feature objects that resemble the ones used by the floor plan generator. Five of them are archived using a ruler for a better precision and two with free-hand drawing to challenge the neural network.

Table 3: Results on hand drawing datasets

Object	mAP	Precision	Recall
Window	0	0	0
Doors	0.840	0.913	0.778
table	0.900	1	0.800
Sink	0.687	1	0.375
couch	0.812	1	0.625
bed	0.750	1	0.500
Toilet	0.642	0.750	0.429
Tub	0.723	0.600	0.750
All	0.669	0.782	0.532

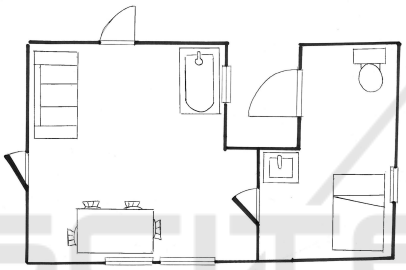

Figure 7: Sample of the Hand-drawn dataset.

Our method yields good results on the hand-drawn dataset. This indicates that the patterns learnt by the neural network during the training is robust enough to understand a hand-drawn version of a floor plan. This results are really interesting to validate our model in a near future. However, a dataset of seven plans is not enough to draw conclusion on the validity of our model on other hand-drawn floor plans.

5 CONCLUSION

In this paper we demonstrated an automated framework to detect and classify objects in floor plans using a neural network trained with a custom floor plan dataset generator. Our custom floor plan dataset generator allows our neural network to be useful on different plan representations. As presented in the section 4.1 and 4.2 the model performs better on synthetic dataset than the real world dataset. The noise and blur induced on realistic floorplan confuses the YOLOv5 instance that has only been fed with synthetic procedural data. A more robust model can be build using a more versatile layout and symbolism on floorplan to generate and including real world datasets in the training process. More variation of the same images with blur and distortion can be added to achieve better results as well.

REFERENCES

Ahmed, S., Liwicki, M., Weber, M., and Dengel, A. (2011). Improved automatic analysis of architectural floor plans. In *2011 International Conference on Document Analysis and Recognition*, pages 864–869.

Chiranjoy Chattopadhyay (2019). Repository of building plans (robin).

Daniel Westberg (2019). Floor plans to blender 3d. https://github.com/grebtsew/FloorplanToBlender3d/.

Davila Delgado, J. M., Oyedele, L., Demian, P., and Beach, T. (2020). A research agenda for augmented and virtual reality in architecture, engineering and construction. *Advanced Engineering Informatics*, 45:101122.

de las Heras, L.-P., Mas, J., Sánchez, G., and Valveny, E. (2011). Wall patch-based segmentation in architectural floorplans. In *2011 International Conference on Document Analysis and Recognition*, pages 1270–1274.

Dodge, S., Xu, J., and Stenger, B. (2017). Parsing floor plan images. In *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, pages 358–361.

Epic Games (2019a). Unreal engine. https://www.unrealengine.com.

Epic Games (2019b). Unreal market.

Foundation, T. B. (2002). Blender. https://www.blender.org/.

Fréville, T., Hamesse, C., Pairet, B., Lahouli, R., and Haelterman, R. (2021). From floor plans to virtual reality. In *2021 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pages 129–133.

Goyal, S., Mistry, V., Chattopadhyay, C., and Bhatnagar, G. (2019). Bridge: Building plan repository for image description generation, and evaluation.

Haas, J. K. (2014). A history of the unity game engine. *Worcester Polytechnic Institute*.

Haelterman, R., Bellemans, M., Lammens, D., Sloover, J., De Vleeschauwer, T., Schoofs, E., Jordens, W., Steenhuyse, B., Mangelschots, J., Selleri, S., Hamesse, C., and Fréville, T. (2020). Training firefighters in virtual reality.

Liu, S., Qi, L., Qin, H., Shi, J., and Jia, J. (2018). Path aggregation network for instance segmentation. *CoRR*, abs/1803.01534.

Macé, S., Locteau, H., Valveny, E., and Tabbone, S. (2010). A system to detect rooms in architectural floor plan images. pages 167–174.

Mathieu Delalandre (2019). Systems evaluation synthetic documents (sesyd). http://mathieu.delalandre.free.fr/projects/sesyd/.

Meta (2012). oculus. https://www.oculus.com/rift-s/?locale=fr_FR.

Park, S. and Kim, H. (2021). 3dplannet: Generating 3d models from 2d floor plan images using ensemble methods. *Electronics*, 10:2729.

Portman, M., Natapov, A., and Fisher-Gewirtzman, D. (2015). To go where no man has gone before: Virtual reality in architecture, landscape architecture and environmental planning. *Computers, Environment and Urban Systems*, 54:376–384.

Reddy, R. R. and Panicker, M. R. (2021). Hand-drawn electrical circuit recognition using object detection and node recognition. *CoRR*, abs/2106.11559.

Redmon, J. and Farhadi, A. Yolov3: An incremental improvement.

Redmon, J. and Farhadi, A. (2017). Yolo9000: Better, faster, stronger. pages 6517–6525.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). 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. MIT Press.

Sharma, D., Gupta, N., Chattopadhyay, C., and Mehta, S. (2018). Rexplore: A sketch based interactive explorer for real estates using building floor plan images. In *2018 IEEE International Symposium on Multimedia (ISM)*, pages 61–64.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

Singh, D. (2019). Object detection in floor plan images.

Timothée Fréville (2022). Datasets for evaluation. https://github.com/Tim-HW/Object-detection-in-floor-plans-for-automated-VR-environment-generation-datasets.

Valve (2003). Steam vr. https://store.steampowered.com/app/250820/SteamVR/.

Varjo (2016). Varjo. https://varjo.com/.

Virtual Bird (2018). Import runtime ue4. https://www.virtualbird.de/ue4Doku/RealTimeImport/.

Wang, C., Liao, H. M., Yeh, I., Wu, Y., Chen, P., and Hsieh, J. (2019). Cspnet: A new backbone that can enhance learning capability of CNN. *CoRR*, abs/1911.11929.

Zeng, Z., Li, X., Yu, Y., and Fu, C. (2019). 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. IEEE Computer Society.

Ziran, Z. and Marinai, S. (2018). Object detection in floor plan images. In Pancioni, L., Schwenker, F., and Trentin, E., editors, *Artificial Neural Networks in Pattern Recognition*. Springer International Publishing.