FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# 3D Photo Mapper

**Juliana Maria Cruz Marques**


Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Miguel Pimenta Monteiro

Second Supervisor: Luís Filipe Teixeira

July 15, 2021

# 3D Photo Mapper

**Juliana Maria Cruz Marques**

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. António Coelho
External Examiner: Prof. Rui Nóbrega
Supervisor: Prof. Miguel Pimenta Monteiro
Second Supervisor: Prof. Luis Filipe Teixeira

July 15, 2021

# Abstract

3D scene understanding from images has been a relevant subject in the last decade in the computer vision field, with many contributions in emerging markets such as robotics, virtual and augmented reality, with applications to Industry 4.0, and real estate. This topic, mainly when applied to an indoor context, includes several tasks: room layout estimation, object detection, and 3D scene reconstruction.

The room layout determines the orientations, positions, and heights of the walls, so that they can be represented as a set of corner positions or edges. In the last years, the main advances in the layout estimation from images are due to the application of deep neural networks since they can automatically learn features and correctly identify their intrinsic concepts. The object detection phase is responsible for detecting the main objects in the input image. This task is one of the most studied ones in the computer vision field. Nowadays, object detection models, based on deep learning, try to obtain increasingly accurate results. The 3D scene reconstruction phase could include selecting a 3D model for each detected object in the object detection phase, estimating their pose, and their respective positioning in the 3D scene. At the end of this phase, a realistic 3D representation of the scene received as input is expected.

This dissertation aims to address the problem of 3D scene understanding, creating a pipeline that enables the 3D reconstruction of the input scene from only one 360º panorama image. This work can later be applied to some specific domain, such as Industry 4.0, by generating an accurate 3D representation of the factory shop floor. This approach will reduce the effort needed to create the virtual factory model and, consequently, provide a more intuitive tool for managers and engineers to supervise and optimize production output.

In this thesis, we propose an original approach of reconstructing indoor scenes, both geometry of the room and objects, using a single 360º panorama image. Our method adapts some of the state-of-the-art techniques of layout estimation and object detection and introduces an approach for the 3D position reasoning of the objects, 3D model selection, and object's pose estimation. In the end, this results in a 3D scene that represents the image given as input. Furthermore, we present a web/desktop application for visualization and adjustment/modification of the final 3D scene generated by our approach.

Currently, there are still few research works that simultaneously address some of the tasks mentioned above. Still, ours is the first one, in our knowledge, to address all these steps, taking advantage of deep learning and using as input a single 360º panorama image. Our pipeline achieved great results in all its sub-tasks and consequently accomplished very realistic and accurate results in generating 3D scenes.

**Keywords**: Indoor scene reconstruction, Layout estimation, Object detection, Panorama images

# Resumo

A compreensão de cenas 3D a partir de imagens tem sido um tópico muito relevante na última década na área da visão por computador, com muitas contribuições em mercados emergentes como na robótica, na área da realidade virtual e aumentada, na Indústria 4.0, e em outros mercados mais convencionais, como a imobiliária. Este tópico, principalmente quando aplicado a cenas interiores, inclui várias tarefas: estimativa do layout de sala, detecção dos objetos e reconstrução da cena 3D.

O layout da sala determina as orientações, as posições e as alturas das paredes, para que a sala possa ser representada como um conjunto de cantos ou arestas. Nos últimos anos, os principais avanços nesta tarefa devem-se sobretudo à aplicação de redes neurais profundas. A fase de detecção de objetos é responsável por detectar os principais objetos na imagem recebida como input. Esta tarefa é uma das mais estudadas na área da visão por computador. Hoje em dia, os modelos de detecção de objetos, baseados em deep learning, tentam obter resultados cada vez mais precisos. A fase de reconstrução de cena 3D normalmente incluí a seleção de um modelo 3D para cada objeto detectado na fase de detecção de objetos, a estimação da sua pose e o seu respectivo posicionamento na cena 3D. No final desta fase, é esperado uma representação 3D realista da cena recebida como input.

Esta dissertação tem como objetivo a criação de uma pipeline que possibilite a reconstrução 3D da cena recebida como input a partir de apenas uma imagem panorâmica. Este trabalho pode posteriormente ser aplicado a um domínio mais específico, como a Indústria 4.0, gerando uma representação 3D precisa do chão de fábrica. Essa abordagem reduzirá o esforço necessário para criar o modelo virtual da fábrica e consequentemente, fornecerá uma ferramenta mais intuitiva para supervisionar e otimizar toda a produção.

Nesta tese, propomos uma abordagem original para a reconstrução de cenas interiores, tanto a nível da geometria da sala como da representação dos objetos, usando uma única imagem panorâmica. O nosso método adapta algumas das técnicas de estado da arte da estimativa do layout e da detecção dos objetos e introduz uma nova abordagem para a determinação da posição 3D dos objetos, da seleção de modelo 3D e para a estimatimação da pose dos objetos. No final, isto resulta numa cena 3D que representa a imagem fornecida como input. Para além disso, apresentamos uma aplicação web/desktop para a visualização e ajuste ou modificação da cena 3D final gerada pela nossa abordagem.

Atualmente, ainda existem poucos trabalhos que abordem simultaneamente algumas das tarefas acima mencionadas. Sendo a nossa aboragem, a primeira, no nosso conhecimento, a abordar todas essas etapas, tirando partido das redes neuronais profundas e usando como input uma única imagem panorâmica. A nossa pipeline alcançou ótimos resultados em todas as suas subtarefas e consequentemente resultados muito realistas e precisos na geração de cenas 3D.

**Keywords**: Indoor scene reconstruction, Layout estimation, Object detection, Panorama images

# Acknowledgements

There are many people who directly or indirectly made it possible for me to finish my master's degree and this dissertation.

First, I am extremely thankful to my supervisor Luís Filipe Teixeira for all the guidance, motivation, and for all knowledge and insights shared with me that improved not only my dissertation and but also my knowledge and passion for discovering new things. I would also like to thank my supervisor Miguel Monteiro for all the suggestions given, which improve my final work.

My second acknowledgment goes to all the people at Critical Manufacturing who have crossed my path and gave me much positive energy and good ideas to improve my work. I want to give a special thanks to Frederico Gonçalves, who guided me more closely at Critical Manufacturing, always with good suggestions and ideas.

My third acknowledgment goes to my boyfriend, Daniel, for his unconditional love and support. Thank you for cheering me up whenever I needed it and reminding me that I can do everything I want.

Lastly, I would like to thank my family, especially my parents, sister, and brother, from the bottom of my heart for supporting me through all my life and for pushing me to be the best I can be.

Juliana Marques

*"The best way to predict the future*
*is to create it."*

President Abraham Lincoln

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| ANN | Artificial Neural Network |
| AP | Average Precision |
| AUC | Area Under the Curve |
| BoF | Bag Of Freebies |
| BoS | Bag of Specials |
| BPTT | Backpropagation Through Time |
| CE | Corner Error |
| CFL | Corners For Layout |
| CNN | Convolutional Neural Network |
| CPS | Cyber-Physical System |
| DPM | Deformable Part-based Model |
| FN | False Negative |
| FP | False Positive |
| FPN | Feature Pyramid Network |
| FCN | Fully Connected Network |
| GC | Geometric Context |
| GPU | Graphics Processing Unit |
| IoU | Intersection Over Union |
| LSTM | Long Short-Term Memory |
| mAP | Mean Average Precision |
| ML | Machine Learning |
| MLP | Multilayer Perceptron |
| OM | Orientation Map |
| PE | Pixel Error |
| RCNN | Region based Convolutional Neural Networks |
| RNN | Recurrent Neural Network |
| ROC curve | Receiver Operating Characteristic curve |
| RPN | Region Proposal Network |
| SPP layer | Spatial Pyramid Pooling layer |
| SPPNet | Spatial Pyramid Pooling Network |
| SVM | Support Vector Machine |
| TN | True Negative |
| TP | True Positive |
| YOLO | You Only Look Once |

# Chapter 1

# Introduction



This chapter introduces this dissertation by contextualizing the problem and presenting its motivations and objectives. Section 1.1 (p. 1) details the context of this problem in the area of 3D scene understanding. Section 1.2 (p. 2) explains the flaws in the current solutions. Section 1.3 (p. 2) describes the problem we aim to solve. Section 1.4 (p. 3) presents the main goals behind this project. Section 1.5 (p. 3) presents one of our pipeline application areas, Industry 4.0, describing its background and the most important concepts related to it. Finally, section 1.6 (p. 5) describes and explains the structure and organization of this document.

## 1.1 Context

3D scene understanding is a relevant yet very demanding topic in the computer vision field. As the name suggests, it is related to extracting knowledge and relevant information about a given 3D scene, which can be represented as an image or a video. This topic, mainly when applied to an

indoor context, includes several tasks: estimation of the room layout, detection of the objects, and 3D scene reconstruction.

The room layout estimation task is responsible for extracting the room's geometry as a set of corner positions or edges, that represent the walls' positions, heights, and orientations. The object detection task is concerned with detecting instances of objects and their respective positions in a given scene. In a 3D scene reconstruction task a realistic 3D representation of the 2D scene is expected as output, often involving previously the two tasks stated above.

In the last decade, there has been a lot of research in 3D scene understanding due to the use of deep learning, leading to many contributions in emerging markets such as robotics, virtual and augmented reality, Industry 4.0, and more conventional ones, like real estate. The 3D scene understanding tasks are described and analyzed with more detail in chapter 3 (p. 25).

This dissertation was proposed by Critical Manufacturing, a software company whose main product is a Manufacturing Execution System (MES), which is a computerized system used in manufacturing, to track and record the transformation of raw materials to finished goods.

> *"MES provides manufacturers in demanding discrete industries a platform for Industry 4.0 success."*
>
> Critical Manufacturing [28]

## 1.2 Motivation

In the context of house indoors, previous works have tried to make a realistic 3D reconstruction of the scene from images using mainly geometric clues to estimate the layout and neural networks for object detection. However, these solutions still need a lot of manual work, mainly for the annotation of the geometric cues in the input images.

Furthermore, currently, no work does a complete 3D scene reconstruction, including the replacement of the detected objects by 3D models automatically and the estimation of the objects' pose and location. Most of the methods in the area of 3D scene understanding are limited to estimating the room's geometry and applying the panoramic image as a texture to the walls and floor of the generated layout.

## 1.3 Problem Definition

The creation of a pipeline to perform a 3D scene reconstruction from just one panoramic image that includes the tasks of layout estimation, object detection, 3D model selection, and object's pose and location would be an innovative and extremely useful tool in the areas of virtual and augmented reality, in real estate, and in industry 4.0.

## 1.4 Goals

This dissertation's main objective is to create a fast and effective process that allows the creation of a realistic 3D scene from a single 360º panoramic image automatically, including the layout estimation, the object detection, and the final 3D scene reconstruction.

As a secondary goal, we expect to integrate our solution in the Critical Manufacturing MES, allowing the visualization and, if needed, the modification of the created 3D scene in the FabLive component.

## 1.5 Industry 4.0

According to the Germany Trade and Invest (GTAI) [40], *"INDUSTRY 4.0 connects embedded system production technologies and smart production processes to pave the way to a new technological age."*

### 1.5.1 Industry evolution

The industry has its growth distributed in different stages, all with great importance. Such stages are represented by Industry 1.0 to 4.0. [15]

- **Industry 1.0** began in the 18$^{th}$ century and was marked by steam power and mechanization of production.
- **Industry 2.0** began in the 19$^{th}$ century by discovering electricity and Henry Ford's assembly line production.
- **Industry 3.0** began in the last decades of the 20$^{th}$ century and was marked by the invention and use of electronic devices, like the transistor and integrated circuit chips, making it possible to automate individual machines to help or replace operators.
- **Industry 4.0** began in the last two decades and utilizes all the principles of previous industrial models, but with higher rates of integration, virtualization, digitization, and technologies.

### 1.5.2 Origins

The term "Industry 4.0" was originated at the Hanover Fair in Germany and has been recognized by other industrial nations as "Connected Enterprise" in the United States and as "Fourth Industrial Revolution" in the United Kingdom. This concept arises as a strategy to soothe the increasing competition from overseas, promoting manufacturing automation and, in consequence, increasing productivity. The German government's idea was to use intelligent monitoring in production processes to assist decision-making and machine maintenance and thus diminish costs and increase German industries' competitiveness [54].

Industry 4.0 is the turning point to the end of the traditional centralized applications and leads to the digitalization era, where everything is digital from business models to production systems.

### 1.5.3 Design principles

According to [108], Industry 4.0 involves six design principles in its model, which are decentralization, virtualization, interoperability, modularity, real-time capability, and service orientation.

**The principle of decentralization** is understood as companies' ability and, consequently, machines and processes to make their own decisions, rather than passing judgments and information hierarchically through central computers. This principle provides more flexibility, enabling operators to reply to changes and adjust whenever necessary. This principle is not only for machines but also for people, as workers in industry 4.0 have greater freedom in decision making.

**The principle of virtualization** is related to creating a virtual copy of the world, which ensures, in case of failure, that all required data, such as security supplies or next work steps, remains available. This principle reduces employees' and teams' time and decision making by providing, sharing, and integrating data virtually in real-time.

**The principle of interoperability** is linked to the cyber-physical system (CPS) that composes smart machines that autonomously exchange information and initiate actions. This principle also ensures a harmonious interaction between humans and machines so that the committed effort is perceived in sync in all the industrial activities.

**The principle of modularity** entails modular systems that can easily adapt to altering requirements by replacing, adding, or removing individual production modules. Thus, production can always adjust to evolving customer demands without lost productivity.

**The principle of real-time capability** ensures that the company has the best answer time to external and/or internal stimuli by analyzing and sharing data in real-time.

**The principle of service orientation** allows the companies' services, cyber-physical systems, and humans to be made available so other companies, cyber-physical systems, or humans can use them. This raises the accessibility of these services and allows the creation of new types of services.

### 1.5.4 Critical Manufacturing MES

In the context of Industry 4.0, arises the need for a visual and intuitive tool that enables managers and engineers to supervise and optimize all processes of the production output of the factory shop floor in a faster way. This necessity could easily be overcome if there was an interactive tool that shows a 3D representation of the factory shop floor as well as all the processes that are happening on each machine in real-time.

This realistic 3D representation of a shop floor would bring many advantages in the manufacturing context, allowing an overview of the factory and enabling better management contributing to reaching Industry 4.0 more quickly through virtualization.

Currently, in the Critical Manufacturing MES, there is a tool that attempts to solve this need, providing a way to create a 3D representation of the shop floor. However, the process of creation of a 3D representation using that tool has some problems: **(a)** a realistic and yet performant 3D scene representation is impossible to create, **(b)** the 3D scene reconstruction is done manually by the implementation teams, making the process very expensive and time-consuming, and **(c)** the entire process is specific for each factory, so it has to be done from scratch.

## 1.6 Document Structure

This chapter introduced the objective of this dissertation, explaining its context, motivation, and the problems it aims to solve. The remaining of this document contains six more chapters and is structured as follows:

- Chapter 2 (p. 7), **Background**, introduces the background and all the information needed to understand this thesis fully.
- Chapter 3 (p. 25), **State of the Art**, describes the state-of-the-art through the analysis of similar or related published works in the domain of 3D scene understanding, involving a critical analysis of the layout estimation, object detection, and 3D scene reconstruction.
- Chapter 4 (p. 41), **Problem and Proposed Solution**, focuses on the current methodologies' issues and limitations, explains the proposed solution, describing in detail the method used in each phase of the proposed pipeline, and presenting the validation methodology to be conducted.
- Chapter 5 (p. 51), **Experimental Setup and Results**, describes step by step the experimental setup and results of each task of the proposed solution, presenting the experiences and the qualitative and quantitative results. Besides, in each phase is analyzed the evaluation process and is demonstrated the developed solution's validation and evaluation.
- Chapter 6 (p. 73), **Application and Solution's Integration**, focuses on the developed Web/Desktop application, presenting the technologies, input and output, and the implemented main features. Besides, it also explains the integration of our solution in the Critical Manufacturing MES.
- Finally, chapter 7 (p. 79), **Conclusions and Future Work**, presents a reflection on the success of this dissertation, providing a synthesis of the main ideas presented and conclusions drawn, and details the difficulties and future work.

# Chapter 2

# Background

This chapter depicts the main concepts needed to fully understand this work's remainder by having a common ground of terminology. Section 2.1 (p. 8) presents the main ideas of Deep learning, going through the basis of artificial neural networks (ANNs) and a more in-depth description of convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Section 2.2 (p. 16) briefly describes the most essential and used evaluation metrics in Machine Learning, focusing on Deep Learning. Section 2.3 (p. 20) does an overview of some of the most used projections in panorama images. In the end, section 2.4 (p. 23) summarizes this chapter.

## 2.1 Deep Learning

Over the last years, with the increase of computational resources, such as the introduction of graphics processing machines (GPUs), and with the easy access to vast datasets, deep learning methods have been shown to outperform previous state-of-the-art machine learning techniques in several fields, with great prominence in computer vision.

### 2.1.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are an attempt to imitate biological neural networks. ANNs can be seen as weighted directed graphs where the nodes are the neurons, and the connections between them are the edges.

There are, according to the information's flow, two main architectural types of ANNs: **feed-forward networks** pass the information forward from input to output, while **recurrent networks** have a feedback loop where information can be fed back into the input at some point before it is fed on for further processing and final output.

The oldest ANN is the perceptron network, presented by Frank Rosenblatt in 1958 [84]. The perceptron has a single neuron and can only solve linearly separable problems. To solve non-linearly problems, a neural network with more than three layers is needed and is called a Multilayer Perceptron (MLP).

A MLP is a feedforward network composed of two or more interconnected layers: an input layer, one or more hidden layers, and an output layer. Figure 2.1 shows an example of a Multilayer Perceptron model [10].

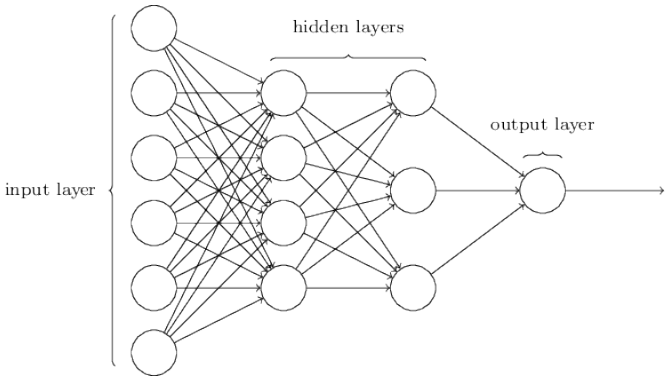



Figure 2.1: An example of Multilayer Perceptron model [10].

#### 2.1.1.1 Weights and Layers

Neural Networks are composed of interconnected neurons (units), which perform weighted sums of their inputs:

$$s_i = w_i^T x_i + b_i, \tag{2.1}$$

followed by a non-linear activation function,

$$y_i = h(s_i) \tag{2.2}$$

as illustrated in figure 2.2.

The $x_i$ are the inputs to the $i$th unit, $w_i$ and $b_i$ are its learnable weights and bias, correspondingly. $s_i$ is the output of the weighted linear sum, and $y_i$ is the final output after $s_i$ is fed through the activation function $h(s_i)$. The outputs of each stage are then fed into units in later stages.

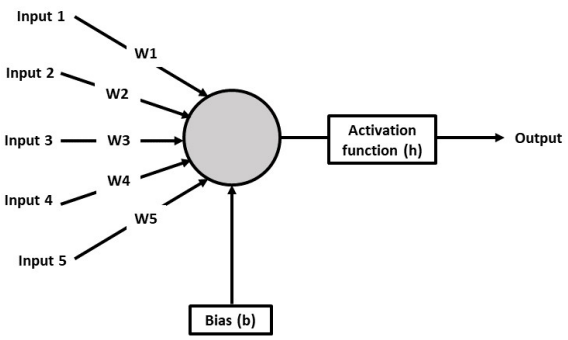



Figure 2.2: An example of a Perceptron model.

#### 2.1.1.2 Activation Function

An activation function is used in an artificial neural network to help the network learn complex data patterns. The activation function takes the output value from the previous layer and transforms it into "something" that can be taken as input to the next layer.

The activation function is also responsible for keeping the value of the neuron bounded to a fixed limit. If not restricted, this value can go very high in magnitude, particularly in deep neural networks with a lot of parameters, leading to computational problems. Another essential feature in an activation function is its capacity to add non-linearity into a neural network, and thus allowing the resolution of non-linear problems. Also, an activation function has to be differentiable; since neural networks are trained using the gradient descent process, the model layers need to be differentiable.

There are, still, more desirable features in the activation functions, such as being zero-centered and avoiding the vanishing gradient problem.

The activation function should be zero-centered therefore the output is symmetrical at zero, and the gradients do not shift to a specific direction.

The vanishing gradient problem appears when we are training artificial neural networks with gradient-based methods, such as backpropagation. This issue makes it hard to learn and tune the earlier layers' parameters in the network and becomes worse as the network's number of layers increases. Gradient-based methods learn by understanding how a small difference in the parameter's value affects the network's output. So, since the gradients control how much the network learns during training, if the gradients are minimal or zero, little to no training occurs, leading to poor predictive performance.

Figure 2.3 presents some of the most used activation functions in deep learning and their main characteristics.

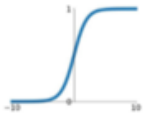
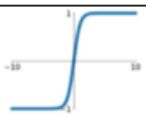
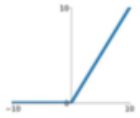
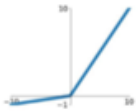
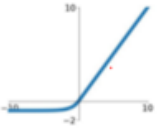
| ACTIVATION FUNCTIONS | FORMULA | GRAPH | COMPUTATIONAL COST | ZERO-CENTERED | DEALS WITH VANISHING GRADIENT PROBLEM |
|---|---|---|---|---|---|
| SIGMOID | $\sigma(x) = \frac{1}{1+e^{-x}}$ |  | High computational cost | No | No |
| TANH | Tanh(x) |  | Low computational cost | Yes | No |
| RELU | Max (0, x) |  | Low computational cost | No | Yes |
| LEAKY RELU* | Max (0.1x, x) |  | Low computational cost | No | Yes |
| ELU** | $\begin{cases} x, x \geq 0 \\ \alpha(e^x - 1)x < 0 \end{cases}$ |  | Medium computational cost | No | Yes |

Figure 2.3: Activation Functions.

#### 2.1.1.3 Loss Function

Loss function measures how "well" our network is at modeling the training examples. If our predictions are poor, the loss function outputs a higher value; if they are reasonable, it outputs a lower value. Loss indicates how much the predicted value differs from the actual value.

There are three types of Loss functions: Regression Loss Functions, Multi-Class Classification Loss Functions, and Binary Classification Loss Functions.

Figure 2.4 presents some of the most used loss functions.

Table 2.1: Loss Functions.

| REGRESSION LOSS FUNCTIONS | BINARY CLASSIFICATION LOSS FUNCTIONS | MULTI-CLASS CLASSIFICATION LOSS FUNCTIONS |
|---|---|---|
| Mean Squared Error Loss | Binary Cross-Entropy | Multi-Class Cross-Entropy Loss |
| Mean Squared Logarithmic Error Loss | Hinge Loss | Sparse Multi-Class Cross-Entropy Loss |
| Mean Absolute Error Loss | Squared Hinge Loss | Kullback Leibler Divergence Loss |

#### 2.1.1.4 Backpropagation algorithm

A neural network learns through a feedback process named **backpropagation**.

This process involves comparing the output a network generates with the output it was meant to generate (ground truth) and using this difference to update the weights of the connections between the layers in the network, going from the output layer through the hidden layers to the input layer, this means going backward in the network.

After a model is designed and hyperparameters are fixed (manually or by a hyperparameter tuning approach), the network automatically learns patterns on the data. This ability is the reason why artificial neural networks are so appealing for researchers and developers.

One of the key properties of neural networks is that they can infer complex concepts out of simpler ones, and that is why even when variance modifies data in unanticipated ways, neural networks can still produce correct predictions.

#### 2.1.1.5 Regularization and Normalization

Regularization and Normalization can prevent neural networks from overfitting and thus better generalize to unseen data. There are several techniques and methods to this end; in this section, we discuss the most used ones.

**Dataset augmentation** aims to reduce overfitting by adding more training samples by perturbing the samples that have already been collected. This technique is particularly effective on image classification tasks since it is expensive to obtain labeled examples, and the image classes should not change under small local perturbations.

**Dropout** randomly deactivates hidden nodes with a fixed probability, in the training phase. This mechanism has the objective of removing connections between neurons that depend on each other. The neuron co-dependency curbs the individual power of each unit and leads to overfitting. As changing weights and nodes during training successfully produces multiple sub-networks, dropout is often thought of as training in parallel many neural networks with different architectures.

**Batch normalization** is a technique that leads to faster learning rates since normalization ensures no activation value is too high or too low and allows each layer to learn independently of the others. To enhance the deep learning network's stability, batch normalization affects the previous activation layer's output by subtracting the batch mean and then dividing by the batch's standard deviation.

### 2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), or simply ConvNets, are a type of deep neural networks that have proven to be very useful in image classification and recognition. CNNs have been extremely successful in facial recognition, diagnostic radiology, understanding climate, and powering vision to self-driving cars and even robots.

One of the first Convolutional Neural Networks with an impact in Deep Learning was the LeNet5 proposed by Yann LeCun in 1998 [61]. At the time, the LeNet5 architecture was mainly used for digit and numeric recognition tasks.

CNNs work in four different phases: convolution, non-linearity, pooling or subsampling, and classification (fully connected layers). Figure 2.4 illustrates an example of a CNN model [17].

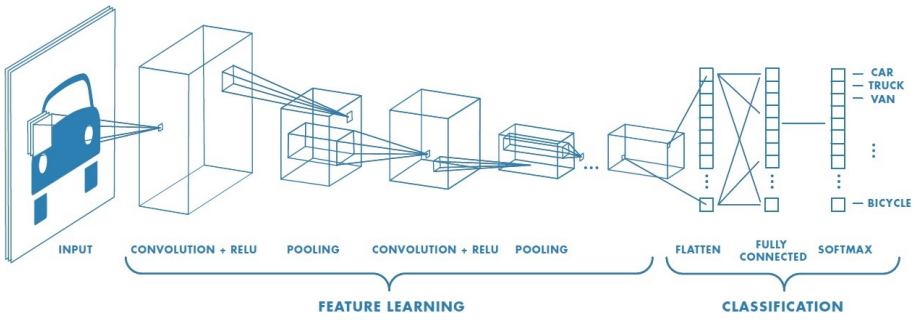



Figure 2.4: An example of a CNN model [17].

A CNN receives an image as input, which will be considered as a matrix of pixel values. A series of convolutions are then performed by sliding a small matrix – filter or kernel – over the input image computing, through the dot product, a feature map. The size of a feature map is influenced by three parameters:

- **stride** – number of pixels by which the filter matrix passes over the image,
- **depth** – the number of filters used for the convolution operation,
- and **zero-padding** – operation of padding the input matrix with zeros around its border.

It is crucial to notice that the filter matrix's different values will generate different feature maps for the same input image.

The operation called ReLu (Rectified Linear Unit) is one type of activation function (section 2.1.1.2 (p. 10)) that can be used after every convolution operation. ReLu is equals to zero for all negative values and linear for all positive values, as shown in equation 2.3. The output feature map after the application of an activation function is called a rectified feature map.

$$f(x) = max(0;x) \tag{2.3}$$

The **pooling step** downsamples rectified feature maps by compacting the most essential features into a smaller matrix. One of the most used functions in this operation is the **max pooling** - extract the largest element from the rectified feature map within that window. Pooling not only reduces the input representation making them more manageable but also minimizes the number of computations and parameters in the network and thus controls overfitting. Overfitting occurs when models are unable to generalize beyond training data.

At this point, the model can understand the features; however, it cannot classify the images. For that purpose, **fully connected layers** are connected to the end of the network. Each neuron on a layer is attached to every neuron on the next layer, as a conventional MLP. The output is fed to a feedforward neural network, and backpropagation is applied to every iteration of training. After a succession of epochs - the number of times a training set is presented, the model can distinguish between low-level features in images and classify them using the standardized activation function: **Softmax**.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}, \tag{2.4}$$

where $x_i$ is the *i*th element of the set of input values.

Softmax takes as input a vector of k real numbers, where k is the number of classes in the problem and normalizes it into [0,1] values proportional to the exponential of the input numbers. The result is a vector with each value in the interval [0,1] and with a sum of all values equals 1, so it can be interpreted as probabilities.

There are many architectures of CNNs available. Some of the most used ones are ResNet, ALexNet, and VGG, as shown in figure 2.5 [5].

The AlexNet architecture was proposed in 2012 by Krizhevsky et al. [59] to compete in the ImageNet challenge, who ended up getting first place. This architecture has 60 million parameters.

In 2015, the VGG architecture was proposed by Simonyan et al.[90] and at the time was considered to be very deep, having 138 million parameters.

Later, ResNet was introduced by He et al. [47] and won first place in several competitions, including the ILSVRC-2015 classification, detection and localization tasks, and COCO-2015 detection and segmentation tasks. This architecture's central idea was the introduction of a residual block, also called skip connections, which add extra connections between nodes in different layers of network that skip one or more layers, making the neural networks dynamic. The skip connections also solve the vanishing and exploding gradient problems.

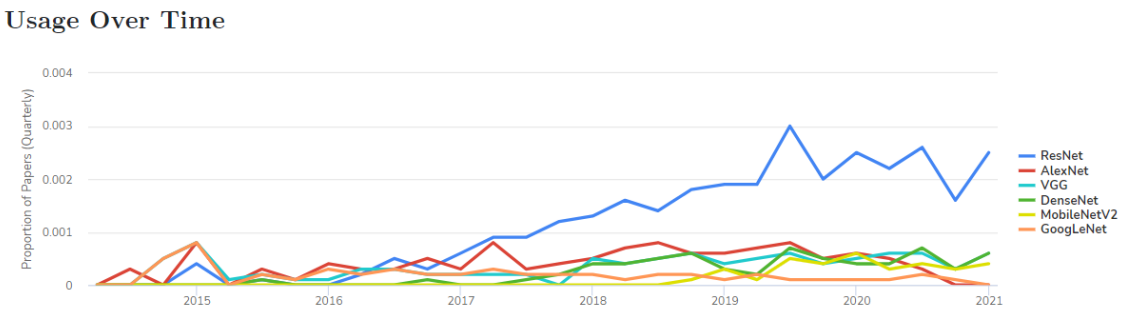



Figure 2.5: Graph of the usage of CNNs architecture over time [5].

### 2.1.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are another category of deep neural networks naturally suited to processing sequential data and time-series data. RNNs have been extremely successful in image captioning, time series prediction, natural language processing, and machine translation.

RNNs were based on David Rumelhart's research in 1986 [85]. Another significant mark in the history of RNNs was the Hopfield networks, discovered in 1982 by John Hopfield. This particular RNN network has all connections symmetric; however, it demands stationary inputs, and because of that, it is not considered a general RNN [51].

There are four types of RNNs, according to their number of inputs and outputs: One to One, One to Many, Many to One, and Many to Many.

- **One to one RNN**, also known as Vanilla neural network, has a single input and a single output.
- **One to Many RNN** has a single input and multiple outputs and is generally used for image captioning.
- **Many to One RNN** takes a sequence of inputs and produces a single output, and it can be used, for example, for sentiment analysis, where a given sentence can be classified as demonstrating negative or positive feelings.
- Finally, **Many to Many RNN** takes a sequence of inputs and produces a sequence of outputs; one of its typical applications is machine translations.

The RNN architectures are represented in figure 2.6 [14].

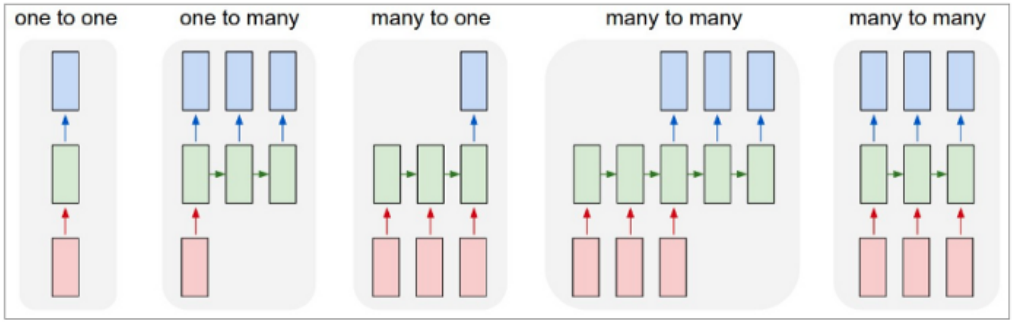



Figure 2.6: RNN architectures [14].

In a RNN, the information cycles through a loop, which allows it to persist the information. The key idea behind them is to consider not only the current input but also the network's hidden state produced in the previous time step, and thus all the inputs are related to each other. Therefore, a RNN has two inputs: the actual and the recent past one, and so apply weights for both through gradient descent and backpropagation through time (BPTT).

**BPTT** is an adaptation of the backpropagation algorithm explained in section 2.1.1.4, which essentially unfolds the network into a feedforward structure and performs the calculations from there in the same way as the backpropagation algorithm [91]. If there are a high number of timesteps, the BPTT can be computationally expensive.

Two major issues prejudice the ability of RNNs to learn long dependencies in time series, which are the vanishing and exploding gradient problems. The error signals computed during backpropagation may become too small or too large with every iteration. With the **vanishing gradients problem**, the weight updates become imperceptible, and thus the learning stagnates. In the case of **exploding gradients problem**, the weight quickly become very large, and so the updates become unstable, which leads to poor results.

In 1997, a solution for this issue was presented in the form of a new recurrent neural network architecture named "Long Short-Term Memory" (LSTM). Alongside this architecture, a modified gradient descent algorithm was also presented. The main difference from BPTT is that the error flow is imposed to be constant throughout the network's internal states [50].

## 2.2 Evaluation metrics used in Machine Learning

Every machine learning model must be evaluated with multiple evaluation metrics, as benchmarks can widely vary based on the selected metric. Furthermore, a model can have a good score in one type of metric but a bad one in another.

### 2.2.1 Classification metrics

#### 2.2.1.1 Confusion Matrix, Accuracy, Precision and Recall

In Machine learning (ML), a confusion matrix is a unique table that enables the visualization of the performance of an algorithm. Each column of the matrix represents the actual class, and each row of the matrix represents the predicted values. Figure 2.7 shows a confusion matrix [4].

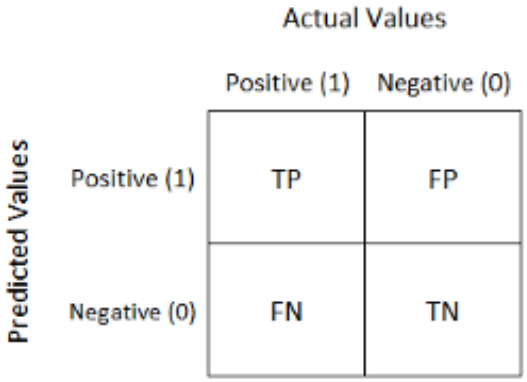



Figure 2.7: Confusion Matrix [4].

**Accuracy** is the proportion of true predictions among all cases. When the problem is unbalanced, it provides highly accurate but purposeless results.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.5}$$

**Precision** is the proportion of true positives (TP) among all positive predicted cases. It shows how likely positive class prediction is correct.

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

**Recall** is the proportion of correctly classified actual positives. Useful when it is a priority to have as many positives as possible.

$$Recall = \frac{TP}{TP + FN} \tag{2.7}$$

#### 2.2.1.2 F1 score

**$F_1$ score**, also known as **F-measure**, is the harmonic mean for precision and recall values, is used when ML researchers attempt to get the best precision and recall values simultaneously. A value close to 1 is synonym of a perfect model; however, a score close to 0 shows a reduction in the model's predictive capability. $F_1$ can also be extended to support multiclass problems and different weights.

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.8}$$

#### 2.2.1.3 ROC Curve and AUC

The **ROC** (Receiver Operating Characteristic) curve plots the graph between the true positive rate (Recall) and the false positive rate (1-recall). The area under the curve (AUC) represents the model predictive capability and is scaled invariant. AUC has a range of [0, 1]. The greater the value, the better is the model's performance. When AUC is 0.5, the model is useless since it is as good as a random model. Figure 2.8 shows an example of a ROC curve [2].

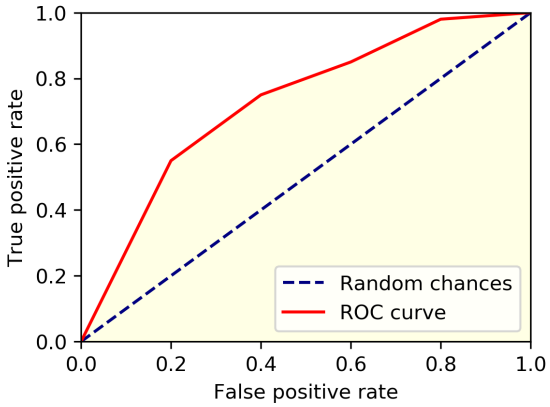



Figure 2.8: Example of a ROC curve [2].

### 2.2.2 Layout Estimation metrics

The most common metrics used to evaluate the layout estimations predictions are:

- **3D Intersection over Union (3D IoU)** is measured between the predicted 3D layout and the ground truth, and then, averaged across all images.
- **Corner Error (CE)** is the L2 distance between the predicted room corners and the ground truth, normalized by the image diagonal, and then, averaged across all the images.
- **Pixel Error (PE)** pixel wise error between the predicted room surfaces labels (ceiling, walls and floor) and the correspondent ground truth, and then, averaged across all images.

#### 2.2.2.1 Intersection over union (IoU)

Intersection over Union (IoU) is an evaluation metric used to calculate the accuracy of an object detector on a certain dataset. To calculate the IoU the ground-truth bounding boxes and the predicted bounding boxes are needed. The IoU is a ratio between the area of overlap between the predicted bounding box and the ground-truth bounding box and the area of the union of both the predicted bounding box and the ground-truth bounding box.

IoU has a range of [0, 1]. The greater the value, the better the prediction. Figure 2.9 shows an example of IoU values for different bounding boxes [6].

$$IoU = \frac{Area\ of\ overlap}{Area\ of\ union} \tag{2.9}$$


Figure 2.9: Example of IoU values for different bounding boxes [6].

### 2.2.3 Object Detection metrics

The most common metrics used to evaluate object detection tasks are the average precision (AP), the mean average precision (mAP), and the intersection over Union (IoU), explained in the previous subsection 2.2.2.1 (p. 18).

#### 2.2.3.1 Average precision (AP) and mean average precision (mAP)

**Average precision (AP)** is defined as the area under the precision-recall curve (PR curve). Figure 2.10 shows an example of a precision-recall curve [13].

Usually, the **mean average precision (mAP)** is computed by taking the mean AP over all classes and/or overall IoU thresholds. However, some papers use AP and mAP interchangeably. For example, for COCO challenge evaluation, there is no distinction between AP and mAP [3]. In the PASCAL VOC2007 challenge, AP for one object class is measure for an IoU threshold of 0.5 [16].

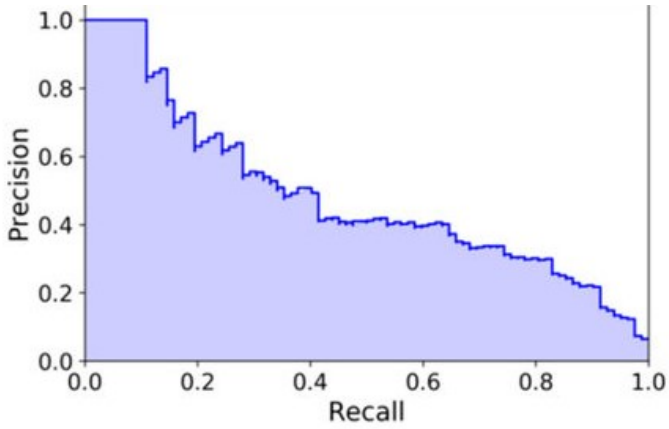



Figure 2.10: Example of precision-recall curve [13].

## 2.3 Panoramic Image Projections

The notion of geometric projection is vital in panoramic photography because it distorts the image and gives it some important context information, essential in many applications, such as in virtual reality games where the user is immersed in that environment.

An image projection happens whenever a flat image is mapped onto a curved surface, or vice versa, and is mainly used in panoramic photography. There are three main types of geometric projections on panoramic images: the straight geometric projection, the curved projections, and the "special" projections [7] [12].

### 2.3.1 Straight geometric projection

The straight geometric projection is commonly called **Rectilinear projection**. This type of projection map all straight lines in three-dimensional space to straight lines on the flattened two-dimensional grid. This type of projection is what most standard wide angle lenses aim to create. Its major disadvantage is that it can significantly amplify perspective as the angle of view grows, causing objects to appear skewed at the edges of the frame. Because of that reason, rectilinear projections are not advised for angles of view greater than 120 degrees.

Figure 2.11 shows an example of a panorama image with a rectilinear projection.

Figure 2.11: Example of a panorama image with a rectilinear projection [7].

### 2.3.2 Curved projections

The curved projections include three main subcategories: the Spherical/Equirectangular projection, the Cylindric projection, and the Mercator projection.

The **Spherical/Equirectangular** projections map a spherical globe's latitude and longitude coordinates directly onto horizontal and vertical coordinates of a grid, where this grid is near twice as wide as it is height. Therefore, horizontal stretching increases farther from the poles, with the south and north poles being stretched across the entire upper and lower edges of the flattened grid. Spherical/Equirectangular projections can display the whole vertical and horizontal angle of view up to 360 degrees, and because of that, they are the most popular for immersive applications.

The **Cylindrical projections** are identical to equirectangular, excluding the fact that they also vertically stretch objects as they get closer to the south and north poles, with infinite vertical stretching happening at the poles. For this reason, this type of projection is not appropriate for images with a large vertical angle of view.

The **Mercator projection** is very well recognizable from its use in the earth's flat maps. This projection provides less vertical stretching and a greater usable vertical view angle than cylindrical projection but with more line curvature. Its most significant disadvantage is that this type of projection exaggerates object's sizes far from the equator (image's horizontal center line).

Figure 2.12 shows the three different curved projections explained previously.


Figure 2.12: Example of panorama images with curved projections [7].

### 2.3.3 Special projections

There are many projection types in the category of the "special" ones. Some of the popular ones are the cubic projection and the little planet.

The **Cubic projection** is a faceted projection consisting of six square sides – front, right, back, left, zenith, and nadir. Each of the 6 square images has a rectilinear projection. Figure 2.13 presents an example of this type of projection.

Figure 2.13: Example of panorama images with cubic projection [11]

The **Litle planet projection**, also called polar projection or stereographic panorama, maps a sphere onto a plane. Figure 2.14 shows an example of this type of projection.

Figure 2.14: Example of panorama images with litle planet projection [9]

## 2.4 Summary

In this chapter, we have created common ground in terms of background knowledge needed to understand the following chapters better. This knowledge comprises an overview of Deep learning, going through the basics concepts of artificial neural networks (ANNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs). Next, an overview of the main evaluation metrics used in machine learning was presented, referring to the standard ones used in the layout estimation and object detection tasks. Lastly, we explained briefly the most used type of projections in panorama photography.

# Chapter 3

# State of the Art



This chapter addresses the different and most relevant studies and methods about scene understanding. Section 3.1 describes the existing methodologies to automatically extract the layout of a room. Section 3.2 introduces the different techniques and methods to detect objects in a scene. Section 3.3 describes the state-of-the-art of 3D scene reconstruction. Section 3.4 describes the most relevant databases of indoor panorama images. In the end, section 3.5 summarizes this chapter.

## 3.1 Layout Estimation

3D layout estimation of an indoor scene from a single view has been a hot research topic during the last decade and plays a crucial role in indoor scene understanding. The idea is going beyond geometrical reconstructions and supply higher-level contextual information about the scene. The layout recovery is behind several emerging applications such as augmented and virtual reality [55], robot navigation [20], scene reconstruction/rendering [98], and real estate [70].

However, layout estimation is not a simple task, and various problems still remain unsolved. Some of the major issues are related to the over-simplification of room types, for example, the assumption of box-shaped layouts. Most of the existing methods are based on strong geometrical assumptions, which often leads to an underfitting of indoor scenes' richness. Also, conventional cameras have a limited field-of-view that causes ambiguities; this problem can be solved by considering wide fields of view, like 360º panorama or fisheye images. Furthermore, there is still the problem of clutter in a room; objects create a challenge to extract key edges and corners that are occluded, making the layout extraction more complex.

In the last years, the most significant advances in layout estimation are derived from the use of 360º panorama images and deep learning, namely deep convolutional neural networks (CNNs). Nevertheless, the application of these entails other problems. For example, deep learning can suffer from lack of data and overfitting, and 360º panorama images are associated with distortions, and thus, it is necessary to adapt or create new methods to solve that efficiently.

### 3.1.1 Related work

Most of the approaches for layout estimation are under the Manhattan World assumption [27]. The Manhattan World assumption imposes that the room layout and even the room's objects are aligned with the three principal axes, that is, all walls are at the right angles to each other and perpendicular to the floor, imposing restrictions on the layout estimation problem.

Delage et al. [32] present, in 2006, a dynamic Bayesian network model to identify the floor-wall boundary in each column of a single perspective image taken by a level camera, and thus, a model capable of recovering a 3D model under the Manhattan World assumption. However, the proposed model needs much prior knowledge about the domain to find the most probable floor-wall boundary in each image.

Lee et at. [64] produce geometric maps through the generation of layout hypothesis based on detected line segments and the posterior selection of the best fitting layout using orientation maps (OM). As well as the previously described, this work assumes that all rooms are under the Manhattan World assumption.

Hedau et al. [49] create a model to generate box layouts by estimating three orthogonal vanishing points, and then the layout is chosen based on edges and geometric context (GC).

Succeeding works follow a similar approach, with advances in the generation of layout hypothesis [87] [88], introducing improved scoring functions [63] [106], and modeling the interaction between objects and layout [31].

All the previously described works use only a single perspective image to estimate the layout, and in consequence of the limited field of view offered by this type of image, usually, they assume a simple box-shaped geometry. An alternative to overcome this issue is to use a wide field of view type of images, such as panorama images that offer a 360º field of view or a fisheye image that offers a field of view in a range of 100º to 180º.

Zhang et al. [104] present the PanoContext dataset and a model to estimate the layout from a single 360º panorama image. The proposed model extends the previous works of hypothesis generation, vanishing points detection, and scoring based on geometric context (GC), orientation maps (OM) and apply them to panorama images.

Li et al. [66] present a model to recover a cuboid-shaped room's layout from a single fisheye image.

Xu et al. [98] present a model, called Pano2CAD that reconstructs a 3D layout of a room, and for the layout estimation also uses orientation maps (OM), geometric context (GC), and object hypothesis applied to panorama images.

The method proposed in [99] uses superpixels, and Manhattan oriented line segments as features and develop the problem through a constraint graph, using panorama images as input. Yang et al.[102] presented a similar method to the previously described one, but with a more complex pre-processing phase: more geometric and semantic cues.

Other approaches tried to estimate the layout on panorama images, using several panorama images [22]. Pintore et al. [78] attempt to extract the layout from a single panorama image using image gradient hints. Zhang et al. [104] estimate indoor scene layouts using RGBD images relying heavily on the sensors' obtained 3D points.

Recent methods started to make use of deep learning to estimate the layout. One of the first methods to take advantage of deep learning was RoomNet [62]. They use deep learning, namely a CNN, to directly predict layout corners in perspective images and a label that points out the room type. They considered eleven room types that represent common cuboid representations under the Manhattan World assumption. RoomNet also uses an RNN to improve 2D corner position predictions.

A year later, the LayoutNet approach appears. Zou et al. [109] predict corner probability map and boundary map directly on panorama using a similar architecture to RoomNet. In contrast to the RoomNet approach, LayoutNet generalizes across perspective images and panoramas, operating directly on the panoramic input, rather than decomposing into perspective images. Furthermore, it can also extend to non-cuboid Manhattan layouts, namely "L"-shaped rooms with good results. They also extend the Stanford 2D-3D dataset [18] with labelled layouts for evaluation and training.

Following the same ideas as in LayoutNet, the HorizonNet method [92] simplifies the LayoutNet's prediction by predicting three 1-D vectors instead of two probability maps. HorizonNet further applies an RNN block to refine the vector's predictions. This approach also applies random stretching data augmentation in training.

Concurrently to the HorizonNet work, Yang et al. [100] present a deep learning framework named DuLa-Net. This framework integrate the surface semantic mask from the equirectangular

view and the projected floor and ceiling view.

In 2020, Fernandez-Labrador et at. [37] present the first end-to-end model that predicts layouts corners, called Corners for Layout (CFL). All the previous works need some pre and/or post-processing steps, such as vanishing point extraction or room model fitting, increasing their computational cost. They propose two alternatives of the network of CFL called EquiConvs and StdConvs. The first one applies equirectangular convolutions directly on the spherical projection, proving to be more robust to camera translations and rotations. The second one applies standard convolutions, reducing the computation time, but needs very sharp images as input.

Table 3.1 below presents some quantitative results of the layout estimation task on the PanoContext dataset, comparing some of the most successful state-of-the-art deep learning methods in this field. Currently, the HorizonNet outperforms all the existing methods in all metrics, in the referred dataset.

Table 3.1: Quantitative comparison of state-of-the-art layout estimation methods

| METHOD | 3D IOU (%) | CORNER ERROR (%) | PIXEL ERROR (%) |
|---|---|---|---|
| PanoContext | 67.23 | 1.60 | 4.55 |
| LayoutNet | 74.48 | 1.06 | 3.34 |
| **HorizonNet** | **82.17** | **0.76** | **2.20** |
| DuLa-Net | 77.42 | - | - |
| CFL | 78.79 | 0.79 | 2.49 |

Figure 3.1 below represents a timeline of the evolution of the layout estimation problem, emphasizing some of its crucial marks. As we can see in the timeline, the PanoContext approach [104], in 2014, was the first method to take advantage of the use of panoramic images. In 2017, with the RoomNet method's publication [62], the introduction of deep learning neural networks in the layout estimation task gained strength. After that, the approaches that have emerged focused more on developing increasingly efficient deep learning neural networks.

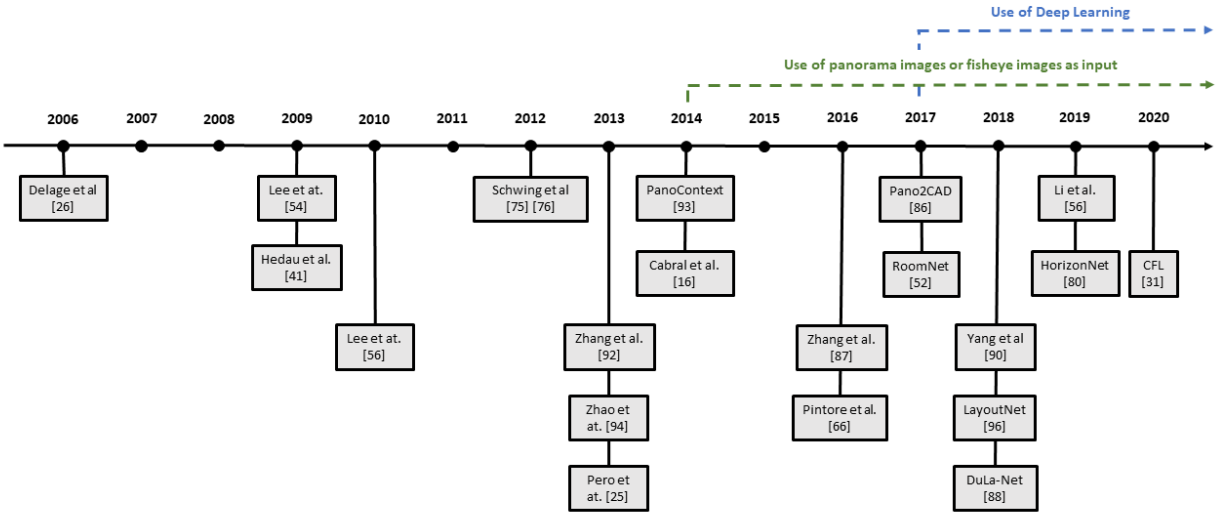



Figure 3.1: Layout estimation timeline.

## 3.2 Object Detection

Object detection is one of the fundamental problems of 3D scene understanding and one of the classic computer vision tasks. Object detection is a technique that enables us to discover where objects are in a given image or video (object localization) and which class each object belongs to (object classification). It can provide precious information for semantic understanding of images and videos related to many applications, including human behavior analysis [30], face recognition [41], autonomous driving [65], and medical imaging [86].

For a human, recognizing a visual concept is relatively easy, and for that reason, sometimes we forget the various challenges that this task entails. Objects in images can present large variations in viewpoints and poses. They can also be occluded for other objects or blended into the environment due to their color or appearance. Besides, they can be affected by different illumination conditions, changing their aspect on the pixel level. Also, the concept behind objects' labels is sometimes broad, having non-clear frontiers to other concepts.

In the last years, the most important improvements in object detection are due to the use of deep learning techniques, namely convolutional neural networks (CNNs), that have already been proved to be the best-known models to do this task, as they can deal with those challenges by automatically learning inherent features of objects and accurately identify their intrinsic concepts.

### 3.2.1 Related work

#### 3.2.1.1 Traditional detectors

The pipeline of traditional object detection models includes three different stages: informative region selection, feature extraction, and classification.

Informative region selection is related to the number of candidate windows/regions to analyze. These regions are produced by scanning the entire image with a multi-scale sliding window. Many candidate regions can be computationally expensive; however, a small number of regions produced by a fixed number of sliding window templates can result in unsatisfactory results.

Feature extraction is related to the extraction of visual features that can provide a semantic and robust representation of different objects. However, because of the diversity of appearances, backgrounds, and illumination conditions, it is not easy to manually design a robust feature descriptor to describe all types of objects correctly. Some of the most used algorithms for this stage are SIFT [75], HOG [29], and Haar-like [67].

Classification is the stage responsible for distinguishing a target object from all the other categories assigning a label. Some of the most popular classifiers are Support Vector Machine (SVM) [26] and AdaBoost [38].

One of the first object detection models was proposed by P. Viola and M. Jones in 2001 [94]. They accomplish real-time detection of human faces for the first time without any restrictions. The detector follows the traditional object detection pipeline, using sliding windows for informative region selection and AdaBoost for feature selection. They also introduced a multi-stage detection paradigm called "detection cascades" to diminish the computational overhead by spending fewer computations on background windows and more on object targets.

Deformable Part-based Model (DPM) was proposed by P. Felzenszwalb et al. [36] in 2008 as an extension of HOG detector [29]. This novel model was the winner of the VOC-07, VOC-08, and VOC-09 detection challenges and so was one of the most famous traditional object detection methods. Felzenszwalb formulates with this model some essential techniques that are still in use and inspire new methods today, such as "hard negative mining", "bounding box regression", and "context priming".

As the performance of hand-crafted features became saturated, discoveries in the object detection field have stopped. In 2014, was proposed Regions with CNN features (RCNN) by R.Girshick et al. [43], initializing the beginning of the era of deep learning in object detection. Object detection can be grouped into two genres: "two-stage detection" and "one-stage detection" in the deep learning era.

#### 3.2.1.2 Deep learning: two-stage detectors

The RCNN model [43] starts with extracting a set of object proposals (candidate boxes) using the selective search algorithm [93]. After that, each object proposal is re-scaled to a fixed size and fed into a convolutional neural network model trained on ImageNet [34] to extract features. In the end, to predict the existence of an object within each region and to assign object classes was used the

linear SVM classifiers. Despite the incredible progress with RCNN, the model had an extremely low detection speed due to the redundant feature computations in many overlapped proposals.

In 2015, K. He et al. presented Spatial Pyramid Pooling Networks (SPPNet) [48]. This model solved the RCNN model's problem by introducing a Spatial Pyramid Pooling (SPP) layer. This layer enables a CNN to produce fixed-length representations regardless of the region of interest size without rescaling it. Thus, the features maps can be produced from the whole image only once, and then, fixed-length representations of regions can be produced. SPPNet is, for that reason, much faster than RCNN. However, SPPNet still had some drawbacks, such as the training is multi-stage, and SPPNet only fine-tunes its fully connected layers while ignoring all previous layers.

Later, R. Girshick proposed the Fast RCNN detector [42]. This detector's most significant contribution was simultaneously training a detector and a bounding box regressor under the same network configurations. Thus, the model reached a better mAP and a much faster detection speed. Although Fast RCNN successfully combines both advantages of SPPNet and RCNN, its detection is bounded by the proposal detection phase.

S. Ren et al. presented the Faster RCNN detector [83] shortly after the Fast RCNN detector. Faster RCNN is the first end-to-end and the first near real-time deep learning detector. This detector's main contribution was the introduction of Region Proposal Network (RPN) that allows almost cost-free region proposals. Furthermore, most detection system blocks, like proposal detection, feature extraction, and bounding box regression, were integrated into a unified end-to-end framework.

In 2017, T. Y. Lin et al. proposed the Feature Pyramid Networks(FPN) [68] based on Faster RCNN. Although the features in deeper layers of a CNN are advantageous for object recognition, before FPN, the detectors run only on the network's top layer. Thus, this model was the first to use a top-down architecture with lateral connections, allowing it to build high-level semantics at all scales.

K. He et al. proposed the Mask R-CNN [46]. This method extends Faster RCNN, adding a branch to predict segmentation masks in a pixel-to-pixel way. Thus, it is one of the first methods to do object detection and image segmentation jointly.

#### 3.2.1.3 Deep learning: one-stage detectors

You Only Look Once (YOLO) was presented in 2015 by R. Joseph et al. and is the first one-stage detector in the deep learning era [80]. As the name suggests, the authors rejected the previous detection paradigm, proposal detection followed by verification, and presented a new paradigm that applies a single neural network to the entire image. This network splits the image into regions and predicts probabilities and bounding boxes for each region at he same time.

Later, the second version of YOLO was proposed by J. Redmon et al. [81]. They focused mainly on improving recall and localization while maintaining classification accuracy. Thus, they added batch normalization on all the convolutional layers in YOLO and a higher resolution classification network, significantly increasing the mAP value. Furthermore, they solved the problem

of only detect one object per grid cell presented in the first version. Thus, this second version can make multi-object prediction per grid cell.

In 2018, J. Redmon et al. presented the third version of YOLO [82]. In this new version, they used a much deeper network, called Darknet-53, with, as the name suggests, 53 convolutional layers. This third version predicts across three different scales. At each scale, the model uses 3 anchor boxes and predicts 3 boxes for any grid cell. Besides, YOLO v3 performs multilabel classification for objects detected in the input image, that is, each object can have more than one label. YOLOv3 performance drops significantly at the IoU threshold. However, it is three times faster than the previous model.

In 2020, A. Bochkovskiy et al. presented the fourth version of YOLO, YOLOv4 [19]. YOLOv4 obtained the current biggest mAP value on the MS COCO dataset and reached a real-time speed of 65 FPS on the Tesla V100, beating all the detectors in terms of both accuracy and speed. They modify the previous model's architecture. The YOLOv4 architecture can be divided into Bag of freebies (BoF), Bag of specials (BoS), and CSPDarknet53. BoF refers to a set of methods that increase accuracy without compromising the hardware, in this case, the most significant additions were the DropBlock regularization and mosaic data augmentation. BoS Refers to a set of modules that only increase the computational cost by a small amount but significantly improve the accuracy, in this case, one of the additions was the Mish activation in the detector.

The Single Shot MultiBox Detector (SSD) was presented in 2015 by W. Liu et al. [72] and was the second one-stage detector in the deep learning era. One of the central contributions of SSD is the introduction of the multi-resolution and multi-reference detection techniques that increases the accuracy of one-stage detectors, namely for small objects. Multi-reference techniques define a set of anchor boxes of different aspect and sizes at distinct image's locations, and then predicts the detection box based on these references. Multi-resolution techniques enable detecting objects at several scales and different layers of the network.

The RetinaNet model was proposed by T. Y. Lin in 2017 [69]. Before this model, despite one-stage detectors presenting high detection speed, they never achieved the two-stage detectors' accuracy. This is mainly because of the extreme foreground-background class imbalance encountered during training. To solve this problem, RetinaNet introduced a new loss function, focal loss. The focal loss reshapes the standard cross-entropy loss so that the detector focuses more on misclassified/hard examples during training. Thus, the RetinaNet detector achieved comparable two-stage detectors' accuracy while maintaining high detection speed.

#### 3.2.1.4 Object detection in panorama images

State-of-the-art approaches mainly focuses on conventional images. However, in the context of 3D scene reconstruction, their limited field of view deteriorates the results, and because of that, images with a large field of view are increasingly used in 3D scene understanding.

Thanks to the increasing research in autonomous vehicles, there are recent works on object detection using 360º outdoor panoramas images [76] [101]. However, there are just a few works on object detection from indoor panoramas.

In 2017, Deng et al. [33] proposed a method to do multilabel object detection in panoramic images using a region-based convolutional neural network (R CNN). They also create a fast method for creating panorama images using three fisheye cameras and presented a new panoramic image dataset for the indoor environment. To deal with the distortion caused by the fisheye images, they applied "longitude-latitude projection", which projects a distorted fisheye image into a square shape as general photos.

In 2020, J. Guerrero-Viu et al. presented an object recognition system that conducts object detection and semantic segmentation applied to panorama images using deep learning [45]. Their model extends the CNN BlitzNet model [35] to deal with panorama 360º images. To deal with this type of image's distortions, they replace all standard convolutions with equirectangular convolutions.

Figure 3.2 represents a timeline of the evolution of the object detectors. As we can see in the timeline, object detectors' evolution can be divided into three main stages. The oldest detectors are categorized as traditional detectors and are characterized by using the sliding window method for region selection. Then, in 2014 emerged the RCNN method [43] and in 2015, the YOLO approach [80], giving rise to Two-stage detectors and One-stage detectors, respectively. The main difference between these types of detectors lies in the fact that two-stage detectors first extract image proposals and then use a neural network to extract features; the one-stage detectors apply a single neural network to the whole image. In dashed lines, we find some approaches developed especially to deal with panoramic images.

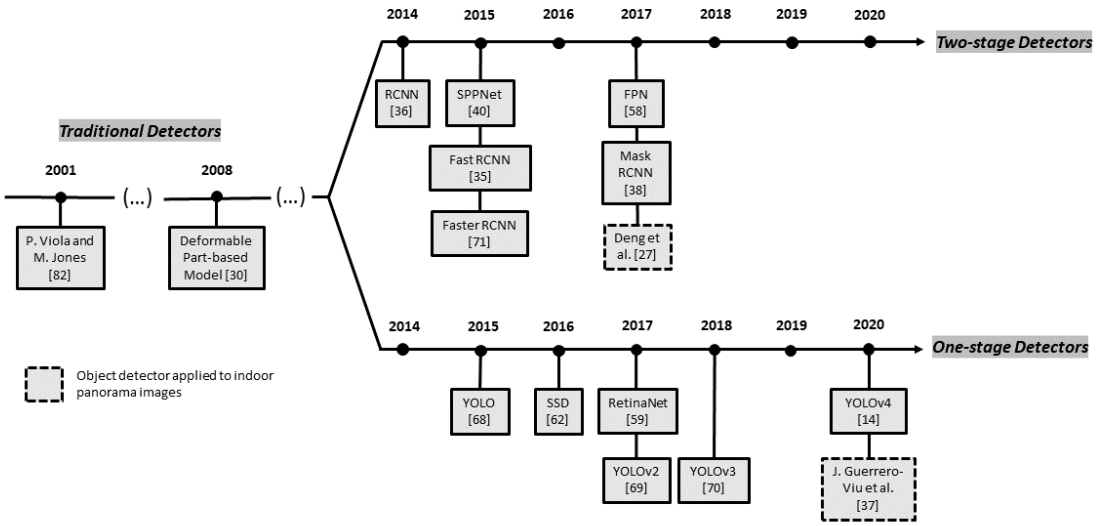



Figure 3.2: Object detectors timeline.

## 3.3 3D Scene Reconstruction

3D reconstruction is a key topic to scene understanding, going beyond 2D analysis. Despite its importance, this task is exceptionally complex, and there are just a few research pieces on this topic, mainly when using panorama images. 3D reconstruction is behind several emerging applications, such as virtual environments and tourism [79], robotic mapping [56], city planning [95], and real estate [70].

3D scene reconstruction from a single image usually involves a set of crucial computer vision tasks, like layout estimation (described in detail in section 3.1), to acquire the geometry of the room, and object detection (described in detail in section 3.2), to obtain the exact position and type of the principal objects in the scene. Beyond that, it is still necessary to estimate the scene object's pose and build the 3D scene, with the 3D objects in their respective positions. Several studies attempt to solve only one of these tasks; however, just a few tried to do the whole process.

Recently, mainly due to the technological advancements in the phone's cameras, capturing panoramic images has become much easier and affordable. The panoramic image usage offers a 360º field of view, and thus a full context of the scene, including the interplay among objects. Therefore, researchers in the field of 3D scene understanding are increasingly using panoramic images instead of perspective images that can only show a little part of the scene.

### 3.3.1 Related work

#### 3.3.1.1 Using an RGBD camera

3D reconstruction using RGBD cameras has evolved drastically in the last years, mainly due to the Kinect sensor launch, which made the use of RGB-D cameras more accessible and affordable.

In 2012, T. Shao et al. [89] proposed a method for semantic 3D modeling of indoor scenes with an RGBD camera . The acquired indoor scene images are first segmented into regions with object labels, and then the segmented objects are replaced by their matched 3D models in a database. This method uses a random forest model as a model instance recognition problem to choose the most alike 3D models in the database to the scene's segmented objects. As the user continues to acquire images of an indoor scene, the system can progressively reconstruct an indoor scene's prototype.

In 2015, Ikehata et al. [52] proposed a 3D modeling approach that reconstructs a 3D indoor scene from panorama RGBD images as a structured model. In this method, the scene's geometry is represented as a graph, where the structural elements, such as walls and objects, correspond to nodes, and their geometric relationships correspond to the edges.

In 2019, Chen et al. [25] presented a novel method called Floor-SP that automatically reconstructs a house's floorplan using RGBD cameras. They perform room segmentation and mapping by using a deep neural network, producing near-perfect results for Manhattan structures. However, this approach only leads with the room layout generation, not considering the object's reconstruction.

#### 3.3.1.2 Using a single perspective image

In 2015, Z. Liu et al. presented a novel 3D indoor scene modeling method using a single perspective image [73]. The 3D scene can be reconstructed using existing model libraries on the internet. The method can be divided into two phases: image analysis and model retrieval. In the image analysis phase, they got the object information from the input image by using geometric reasoning combined with image segmentation. In the model retrieval phase, line drawings were extracted from 2D objects and 3D models using distinct line rendering algorithms. They used several tokens to express local feature and then organized it as a star-graph. In the end, by comparing the similarity among the encoded line drawings, models were retrieved from the model library, and the 3D scene was reconstructed.

In 2017, H. Izadinia et al. proposes an entirely automatic approach named IM2CAD, which, from a single perspective photo of an indoor room and an extensive furniture CAD model database, reconstructs a 3D scene similar as possible to the scene described in the input image [53]. The presented model includes several steps, including room geometry estimation, object detection, CAD model alignment, object placement in the scene, and scene optimization via render and match.

They used an end-to-end deep Fully Convolutional Network (FCN) that estimates per-pixel surface labels for room geometry estimation. To detect the main objects in the input image, they used Faster RCNN. To determine the objects' shape and their approximate pose, they rendered each 3D model of the ShapeNet repository [24] into 32 viewpoints, and then using a CNN, they computed the features for each of the rendered images and for each detected bounding box. The best rendering match is the one with a higher value of cosine similarity, and thus the CAD model and its orientation are chosen. To do the objects' placement in the room geometry estimated, they first estimate the camera rotation and the intrinsic camera parameters concerning the room space using three orthogonal vanishing points and choose one of the visible room corners as the origin of the coordinate system. Finally, they refine the objects placements in the scene by optimizing the visual similarity between the rendered scene and the input image.

#### 3.3.1.3 Using a single panorama image

In 2014, proposed by Y. Zhang arises the PanoContext method [105]. This method outputs a 3D cuboid room layout with the detected scene objects represented by their 3D bounding boxes. Their method consists of two steps: bottom-up hypotheses generation and holistic hypotheses ranking.

To generate hypotheses, they first estimate vanishing points based on the detected line segments. Then, they generate 3D room layout hypotheses from line segments and verified them with the calculated geometric context (GC) and orientation map (OM) on the panorama. For objects, they generated 3D cuboid hypotheses using rectangle detection and image segmentation. Next, they used sampling to generate whole-room hypotheses, each with a 3D room and multiple 3D objects inside. To choose the best hypothesis coherent with the input image, they extracted various features and trained an SVM to rank these hypotheses holistically. In the end, they locally

adjusted the best hypothesis and searched for a solution that maximizes the SVM score by adding, swapping and deleting an object.

Furthermore, jointly with the PanoContext method, they presented an annotated indoor panorama dataset.

In 2017, J. Xu presented a new 3D scene reconstruction method from a single indoor 360º panorama image, called Pano2CAD [98]. This method integrates normal surface estimation, 2D object detection, and 3D object pose estimation.

First, they transform a single panorama image into a set of perspective images, from which they estimate per-pixel surface orientations and do the object detection. To determine the room's surface orientation, they estimated their Orientation Map (OM) and Geometric Context (GC). Then, they applied GC to the panorama image and combined it with the OM and GC in the floor region to get wall orientations and positions. To perform object detection, they used the Faster R CNN model. In the end, to estimate the 3D object pose, they collected a set of 3D models from the 3D Warehouse [1] and rendered each in 360 poses. TRW-S [58] was run for 100 iterations to estimate the object's pose.

In 2020, W. Zeng et al. proposed a novel method for 3D indoor semantic scene point cloud from a single panorama image, called Pano2Scene [103]. This method combines layout estimation and generation of objects' point cloud in the input image. First, to obtain the scene's 3D layout, the Pano2Scene method estimates the layout depth map and reconstructs the parameterized 3D layout. Second, to produce the objects' full point cloud in the scene, the method finishes the object point cloud from the visible partial point cloud via global feature vector mapping. Finally, to impose consistency between the reconstructed scene and the panorama input image, the algorithm projects the inferred complete object point cloud back to the 2D panorama and jointly trains the full pipeline end-to-end.

#### 3.3.1.4 Comparison of 3D indoor scene reconstruction methods

Table 3.2, presented below, summarizes the state-of-the-art methods of 3D indoor scene reconstruction. This table compares all the previously described 3D indoor scene reconstruction methods regarding the type of input image and procedure used to do the layout estimation, object detection or segmentation, and 3D model placement/choice. The following table presents the approaches in chronological order, beginning with the older previously analyzed one.

As we see through the analysis of table 3.2, no approach applies deep learning in the layout estimation and object detection phase simultaneously. Pano2CAD [98] and IM2CAD [53] are the approaches that most benefit from deep learning taking advantage of it in the object detection phase, and in the case of IM2CAD, also to compute features for each of the 3D models in the 3D objects placement/choice.

Table 3.2: State-of-the art methods comparison of indoor 3D scene reconstruction.

| METHOD | INPUT IMAGE | LAYOUT ESTIMATION | OBJECT DETECTION/SEGMENTATION | 3D OBJECTS MODEI'S PLACEMENT/ CHOICE |
|---|---|---|---|---|
| **T. Shao et al. [89]** | A single/various RGBD images | - | A novel context-aware image segmentation algorithm. | Use random regression forest for model matching. |
| **PanoContext [105]** | A single panorama image | Generates hypothesis for room layout and object segmentation using evidence, such as edge, segmentation, normal direction estimation, OM, and GC. SVM ranks the 3D scene hypothesis and chooses the best one. | | - |
| **Z.Liu et al. [73]** | A single perspective image | Use geometric reasoning. | Use a proposed image segmentation method. | Comparison of the segmented objects' line drawings and the library models' line drawings. |
| **Ikehata et al. [52]** | A single/various RGBD panorama images | the scene's layout is represented as a graph based on geometric rules | - | is used a point-cloud as object geometric representation |
| **IM2CAD [53]** | A single perspective image | Learns geometry from hand-designed low-level descriptors, like color and texture. | Faster R-CNN | Use a Faster R-CNN to compute deep features for each of the 3D model's rendered images and the detected image bounding box and use cosine similarity as the distance metric. |
| **Pano2CAD [98]** | A single panorama image | Estimate per-pixel surface orientations, combining estimates of their OM and GC. | Faster R-CNN. | For each object detected, they search for the k-nearest neighbors among a set of rendered images in a 3D database. |
| **Pano2Scene [103]** | A single panorama image | Obtain the layout depth map and reconstructs the parameterized 3D layout. | To produce the objects' full point cloud in the scene, the method completes the object point cloud from the visible partial point cloud via global feature vector mapping. | - |

## 3.4 Relevant Datasets

### 3.4.1 Indoor panorama scene's datasets

Datasets with a detailed ground truth have a crucial role in both network training and performance validation. The most used public datasets of indoor panorama images are PanoContext [105], Stanford 2D-3D [18], Matterport3D [23], and SUN360 [97].

All four datasets are composed of RGB panorama images of different types of rooms, like "L-shaped" rooms, cuboid rooms, and "T-shaped" rooms. They differ in the layout's complexity, in the diversity of scene types, and in the respective dataset scale.

The PanoContext dataset [105] contains 552 RGB panorama images of two types of indoor rooms, which are living rooms and bedrooms. In this dataset, all the images are annotated as cuboid layouts.

Stanford 2D-3D dataset [18] contains 552 RGB panorama images collected from 6 different indoor environments, capturing 2D, 2.5D, and 3D data with instance-level semantic and geometric annotations. Unlike the PanoContext dataset, this dataset represents much larger scenes like offices, classrooms, and other open spaces, such as corridors. The original data does not provide ground truth layout annotations. The Stanford dataset is more demanding than the Panocontext one because the images have more occlusions on the wall-floor boundaries, and the vertical field-of-view in this dataset is smaller than in the PanoContext dataset.

Matterport3D [23] is an enormous RGBD dataset containing over ten thousand RGBD panoramic images collected from ninety building-scale scenes. Matterport3D dataset has the advantage of covering a bigger variety of room layouts and types of environments than the previously described datasets. Besides, it is three times larger than the Stanford and PanoContext datasets, thus providing richer data for training and evaluation. The precise global alignment and multiple panoramic views over whole buildings allow keypoint matching, view overlap prediction, semantic segmentation, and region classification.

SUN360 is a large dataset of indoor and outdoor panoramas [97], with 369 types of environments and 10,335 RGB-D images from four different sensors with dense labels, and it also comprises two-dimensional polygons and 3D bounding boxes with accurate object orientations, as well as a 3D room layout and scene category for each image. The original data does not provide ground truth layout annotations.

#### 3.4.1.1 3D model databases

3D Warehouse is an extensive 3D model database with objects of different types of environments, including indoor ones [1].

ShapeNet [24] is a large CAD model database with various objects, including typical indoor objects, similar to the 3D Warehouse.

## 3.5 Summary

This chapter introduced the principal research areas that support this dissertation.

In section 3.1 (p. 26), we started by explaining what layout estimation is and its importance for 3D scene understanding, then the main studies in this area were analyzed and described in detail, concluding that the current dominant approaches are HorizonNet [92], CFL [37], and DuLa-Net [100].

In section 3.2 (p. 29), we explained the main objectives for object detection and why this field is crucial in 3D scene understanding. Then, we explained object detectors' evolution in detail, identifying three different stages of approaches: traditional detectors, two-stage detectors, and one stage detectors. Besides, we also analyzed some approaches that have as input only panorama images. We concluded that currently, the most used approaches are the YOLO [80, 81, 82, 19] and the Faster RCNN [46].

In section 3.3 (p. 34), we described and analyzed all the models found, with the primary objective of the 3D reconstruction of an indoor scene. We conclude that there are just a few works on 3D indoor scene reconstruction and even fewer works dealing with panorama images, and to our knowledge, there is no 3D scene reconstruction work that involves deep learning in their sub-tasks.

Finally, in section 3.4 (p. 38), we presented and described the most-used datasets of indoor panoramas, as well as some databases of 3D models.

# Chapter 4

# Problem and Proposed Solution



This chapter describes the problems detected by analyzing the state-of-the-art solutions that this work aims to tackle and presents a proposed methodology to solve them, explaining in detail all the steps needed to follow the proposed solution pipeline and thus achieve the central goal of this dissertation.

Section 4.1 (p. 42) describes the limitations present in the current state-of-the-art approaches that remain to be addressed. Section 4.2 (p. 42) gives an overview of this project solution, explaining all the tasks lightly. The following four sections in this chapter correspond to an essential step in the proposed solution pipeline. Section 4.3 (p. 44) explains the method used in the proposed solution's first step: **Layout Estimation**, describing the approach and the needed assumptions.

Section 4.4 (p. 45), similarly to the previous section, describes the method used in the proposed solution's second step: **Object Detection**. Section 4.5 (p. 46) describes the details of the proposed solution's third step: **3D Position Reasoning**, which starts by affirming the necessary assumptions and explaining the used method. Section 4.6 (p. 49) depicts the last crucial step in the proposed pipeline: **3D Model Selection and Object's Pose Estimation**, which details the used method. Section 4.7 (p. 50) details the experimental methodology. Finally, section 4.8 (p. 50) summarizes this chapter.

## 4.1 Current Issues

Despite the significant advances in 3D scene understanding, nowadays there are still very few works that include a complete 3D scene reconstruction and even fewer using panorama images as input. Besides, the few works that attempt to do a 3D reconstruction of a scene use methods that require many pre- and post-processing steps, such as the indication of geometric or segmentation cues.

Regardless of the incredible results in the layout estimation and object detection tasks, mainly due to the application of deep learning networks, no one, to our knowledge, has yet tried to take advantage of these improvements to create a novel approach based on deep learning that is able to reconstruct a 3D scene from a panoramic image.

## 4.2 Proposed Methodology

As explained in the previous chapters, the main objective of this dissertation is the development of a pipeline that allows the creation of a 3D scene based only on one panorama image. For that purpose, we need to estimate the room's geometry, detect the main objects in the scene, estimate the object's pose, location, and size, select a valid 3D model for each of the detected objects and also create a tool that enables the visualization, and whenever needed, the modification of the created 3D scene.

To fulfill the objectives, our approach has two critical parts. The first one was developed in Python and is responsible for the JSON file generation with all the necessary data to reconstruct a 3D scene. This phase includes the layout estimation, the object detection, the 3D position reasoning, the 3D model selection, and the object's pose estimation. The second part of our approach was mainly developed in JavaScript and is responsible for the visualization of the 3D scene based on the JSON file generated in the first part. This phase includes the web/desktop application development and comprises the posterior integration in the Critical Manufacturing MES.

Figure 4.1 illustrates the solution's diagram.

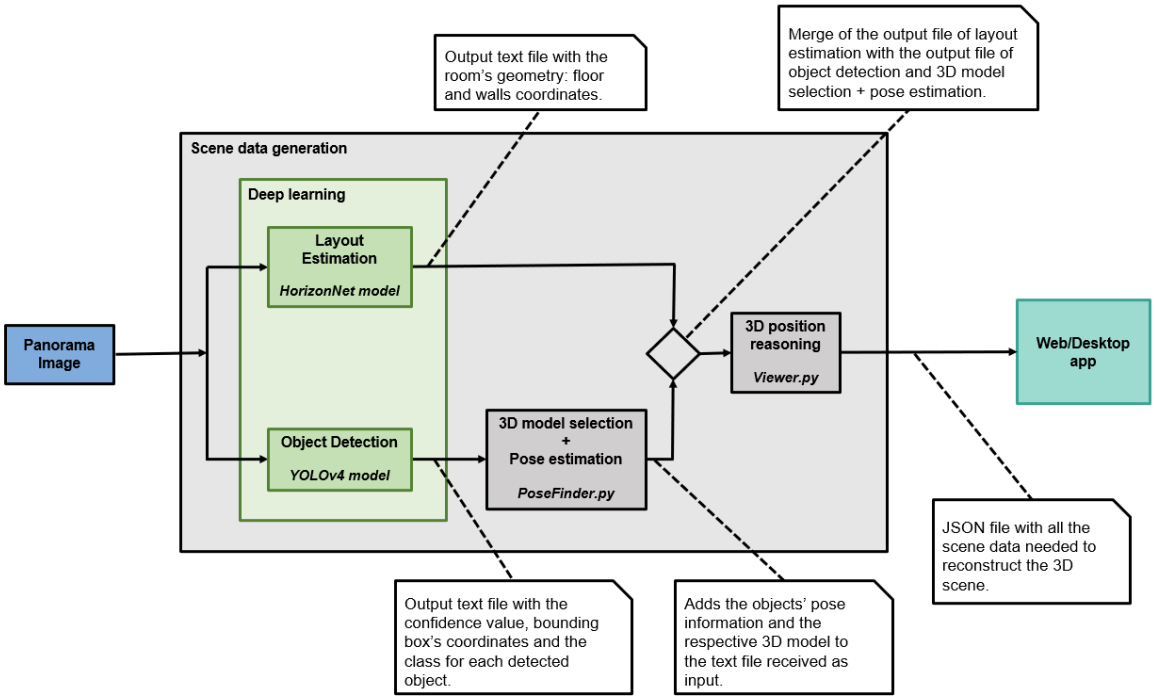



Figure 4.1: Solution's diagram.

According to the diagram in figure 4.1, initially, the input panorama photo feeds the HorizonNet model [92] to estimate the room's geometry and feeds the YOLOv4 model [19] to detect the main objects in the image. Both models generate a text file with their respective results. The HorizonNet's text file output contains the coordinates of the floor and walls. The YOLOv4 text file output includes a list of the detected object's class and their respective bounding box coordinates and confidence value.

Then the YOLOv4 text file feeds the *PoseFinder* Python module that will select a 3D model to represent each of the objects present in the input file and their respective pose, adding this new information to the YOLOv4 text file.

Subsequently, the *Viewer* Python program receives as input the HorizonNet's text file and the resulting text file of the 3D model selection and pose estimation. The *Viewer* Python program estimates the object localization within the room's geometry and generates the final output, a JSON file, that will provide all the necessary information for the web/desktop application to generate the 3D scene.

The following sections will describe in detail each of the steps stated above and represented in figure 4.1.

## 4.3 Layout Estimation Task

As explained in chapter 3 (p. 25), the layout estimation has the main objective of extracting the edges and corners of an input image, thus recovering its room structure (walls, ceiling, and floor).

To estimate the 3D room layout from a single panorama image, we employed the HorizonNet method [92]. To train, test, and validate the model we used the Google Colab[1], a hosted notebook service that provides free access to computing resources, including GPUs.

### 4.3.1 Method

For this step, we propose to employ the HorizonNet approach [92] since, as we see in section 3.1 (p. 26), it is currently the Layout estimation method with the best results. Their network comprises two main parts: a feature extractor and a recurrent neural network, followed by a post-processing step. Figure 4.2 illustrates the HorizonNet architecture [92].

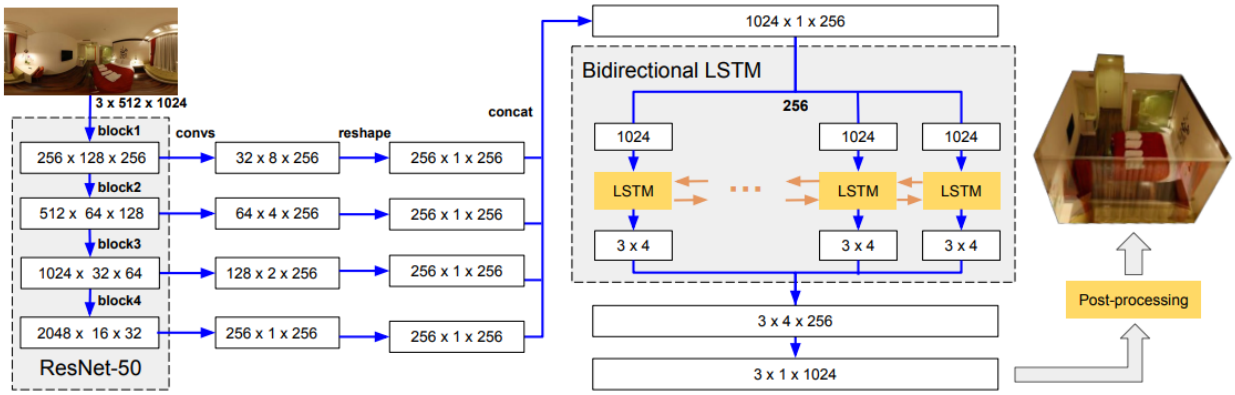



Figure 4.2: HorizonNet architecture [92].

**The feature extractor** used was the ResNet-50 [18]. To capture low- and high-level features, each block of the extractor has three convolution layers in which a factor of 8 and 16 reduces the number of channels and the height, respectively. All the extracted features from each layer are upsampled to the same width and reshaped to the same height. The activation function used after each convolution layer is ReLU except for the final layer where the Sigmoid function is applied.

**The recurrent neural network** receives as input the final concatenated feature map of size $1024 \times 1 \times 256$. The bidirectional LSTM architecture [50] is adopted. RNN is used because it is capable of learning patterns and dependencies from sequential data, storing information about its predictions in the internal state, and so it can predict accurately for occluded area based on the entire room's geometric patterns. This phase's final outputs represent the layout information for each image column ($y_{floor}$, $y_{ceiling}$, $y_{wall}$ ). The $y_{floor}$, $y_{ceiling}$, and $y_{wall}$ represent the probability of

[1] https://colab.research.google.com/

floor-wall boundary, ceiling-wall boundary, and wall-wall boundary at the corresponding image column, respectively.

**In the post-processing step**, we recover the room geometry based on the probability image columns provided from the previous phase. Knowing that the ceiling-wall boundary shares the same 3D (X, Z) position with the floor-wall boundary on the same image column, we can determine the ceiling-floor distance based on the assumed camera height. So, the average of the ceiling-floor distance on each image column is the ceiling-floor distance (Y). To recover the planes of the walls, we find the prominent peaks on the estimated wall-wall boundaries with the criteria that the signal must be larger than any other signal within 5º H-FOV and at the same time be larger than 0.05. Then, each peak vote for all planes within 0.16 meters. The most voted plane is selected.

At the end of this phase, it is expected to have an indoor scene model building, where the walls and floor are the segmented areas of the panorama image.

### 4.3.2 Assumptions

In order to estimate the layout of the rooms accurately, some assumptions had to be made.

**Assumption 1.** *Intersecting walls must be perpendicular to each other - Manhattan world assumption [27];*

**Assumption 2.** *All rooms must have their floor and ceiling parallel to each other*;

**Assumption 3.** *The panorama photo must have a spherical/equirectangular projection*;

This type of projection offers essential distortions for the layout estimation.

**Assumption 4.** *The panorama photo must be taken at a height between 1.6 to 1.7m.*

## 4.4 Object Detection Task

As explained in chapter 3 (p. 25), the object detection task is responsible for classifying and localize objects in an input image.

For this step, we propose to take advantage of the YOLOv4 approach [19] since, as we see in section 3.2 (p. 29), it is currently the Object Detection method with the best results. To train, test, and validate the model, as well as in the previous phase, we take advantage of the Google Colab service[2].

### 4.4.1 Method

As the modern detectors, the YOLOv4 network [19] comprises three parts: a Backbone, a Neck, and a Head. Figure 4.3 illustrates the YOLOv4 architecture.

[2]https://colab.research.google.com/

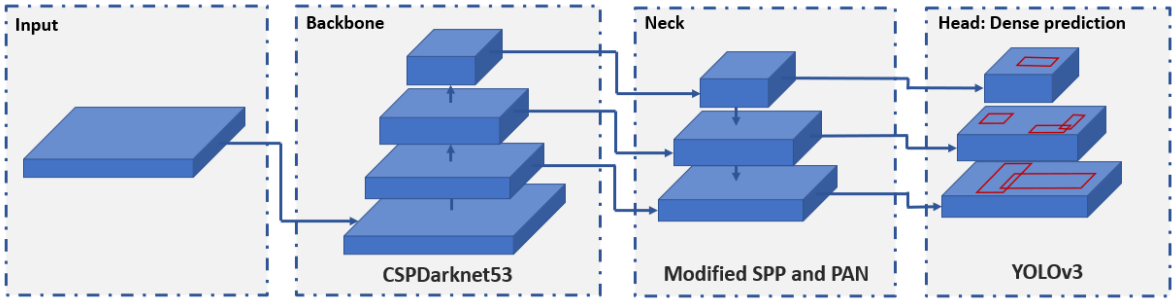



Figure 4.3: YOLOv4 architecture.

**The backbone** refers to the feature extraction, in this case, is pre-trained on ImageNet [34]. They choose the CSPDarknet53 [96], with 53 convolutional layers.

**The neck** is used to add extra layers between the backbone and the head, and so collects feature maps from different stages. They applied a modified version of SPP [48] and a modified version of PAN [71].

**The head**, in this case, a dense prediction, has the objective of predicting classes and the object's bounding boxes. In this model, they used the previous detector, YOLOv3 [82].

The loss function used in the YOLOv4 method [19] with the best results is the Complete Intersection over Union (CIoU) [107]. This loss function incorporates all geometric factors, taking into account not only the intersection over union (IoU) but also the normalized central point distance and the aspect ratio. These factors help the function to convergence faster and to achieve a better performance.

At the end of this phase, it is expected to have all the input image's main objects' labels, including the coordinates of the objects' bounding boxes and their respective classes.

## 4.5 3D Position Reasoning Task

The main objective of this task is to estimate as accurately as possible the 3D location of the objects previously detected in the preceding phase by the object detector and thus position them within the estimated room's geometry obtained in the layout estimation phase.

For this, a Python program named *Viewer.py* was created. It receives as input a text file, which corresponds to the combination of the output of the first step, layout estimation task, the output of the second step, object detection task and also the output of the 3D model selection and object's pose estimation task that will be discussed in the next section.

Figure 4.4 represents an example of the input text file of 3D position reasoning Python program.

```
img path:demo_aligned_rgb.png
floor color:6f321a
walls color:c9ac6d
dimentions:[2.9097135, -1.6, -4.0935926]
floor coord:[[-0.1959936   2.0545576 ]
 [-0.19599354  3.9839153 ]
 [-1.3569014   3.9839144 ]
 [-1.3569022  -4.093592  ]
 [ 2.9097128  -4.0935926 ]
 [ 2.9097135   1.1095303 ]
 [ 1.0311482   1.1095303 ]
 [ 1.0311477   2.0545578 ]]
objects:
Bedroom-chair: 18% (left_x:   -1   top_y:  119   width:    7   height:   47   rot: 0)
Bedroom-chair: 98% (left_x:   25   top_y:  109   width:   28   height:   32   rot: -90)
Bedroom-chair: 95% (left_x:  135   top_y:  111   width:   40   height:   47   rot: -90)
Table: 99% (left_x:  140   top_y:  108   width:   70   height:   28   rot: 90)
Bed: 100% (left_x:  212   top_y:  106   width:   95   height:   81   rot: 90)
```

Figure 4.4: Example of the input text file of 3D Position Reasoning task.

### 4.5.1 Method

Initially, the idea was to take advantage of deep neural networks to estimate the depth in an image. However, despite numerous studies on outdoor photos, namely traffic scenarios using the KITTI dataset [39], in our knowledge, there is no indoor dataset annotated for depth estimation, and there is no model that had been trained with indoor images too.

Despite that, we experimented estimate the depth using the model Monodepth2 proposed by Godard et al. [44] with some of the images in our dataset. Still, the results were meaningless due to the enormous differences between the training images from the KITTI dataset [39] and the indoor panoramic images used for testing.

Training the model with indoor images became impossible due to the lack of a dataset with the required labels. The process of annotating a set of images would also be unfeasible, as it would be necessary additional information such as the real measurements of the objects represented in the image (in meters), which is not available. Thus, the idea of taking advantage of deep neural networks to estimate the depth had to be discarded.

The method adopted was inspired by the methodology used in the HorizonNet approach [92] to convert the pixel coordinates of the corners to 3D coordinates.

This approach takes advantage of the characteristic distortions of the panorama images with spherical/equirectangular projection to determine the 3D coordinates of the detected objects. The main steps are:

1. Calculation of the bounding box centroid (w, h) for each object detected previouly, *where w* $\in$ *[0, 1024] and h* $\in$ *[0, 512].*

2. Representation of each object's centroid coordinates under UV space (u, v), *where u* $\in$ *[-*$\pi$*,* $\pi$*], and h* $\in$ *[-*$\pi$*/2,* $\pi$*/2].*

3. Computation of the 3D world coordinates from the UV space, assuming a height of 1.65 m when the photo was taken (x, y), *where* $x \in [-\infty, +\infty]$, *and* $y \in [-\infty, +\infty]$.

4. Application of a scale adjustment so that all objects are within the previously calculated layout (x, y), *where* $x \in$ *[min x layout, max x layout ], and* $y \in$ *[min y layout, max y layout].*

5. Calculation of the objects' size in x, y, and z considering their respective bounding box measures and assuming a cubic volume.

At the end of this phase, a 3D reconstruction of the input scene with all its principal objects in the right locations is expected. Although, the reconstruction results may need some posterior manual adjustment.

### 4.5.2 Assumptions

To ensure the success of this task, some assumptions have to be established.

**Assumption 1.** *The image's aspect ratio used to estimate the layout must be the same as the image used to detect the objects*.

Thus, the coordinates of the detected objects' bounding boxes are at the same scale as the coordinates of the room's corners. Consequently, the object's coordinates and room walls' coordinates are comparable.

**Assumption 2.** *The panorama photo must be taken at a height between 1.6 to 1.7m.*

In the object position reasoning, as well as in the conversion of layout image coordinates to 3D world coordinates, we assume a camera height of 1.65, but this can vary between 1.6 to 1.7m.

**Assumption 3.** *The panorama photo must have a spherical/equirectangular projection.*

This type of projection offers essential distortions for the 3D position reasoning since it can be projected to the sphere, enabling depth estimation.

**Assumption 4.** *Only objects previously detected by the object detector can be represented.*

The object positioning reasoning program will only receive as input the bounding boxes of previously detected objects, so only these can be represented later in the 3D scene.

**Assumption 5.** *The object's size is calculated considering the bounding box's width and height, and its depth is estimated assuming that the object occupies a cubic volume.*

Due to the lack of information on the objects' real size, with only the panorama image where they are inserted as data, it was necessary to assume a cubic volume for all objects, therefore making the estimation of their size more accurate.

**Assumption 6.** *All objects are within the layout generated in the previous stage of layout estimation.*

As well as all objects are within the room in the panorama image, in the 3D scene representation, all objects have to be within the room's walls.

## 4.6 3D Model Selection and Pose Estimation Tasks

To complete the reconstruction of the scene in 3D to be carried out successfully, it is then necessary to do the automatic selection of 3D models to represent the objects and the choice of their pose. The object's pose refers to the angle that it makes relative to the camera.

### 4.6.1 Method

Since the use of models to estimate depth are not possible to apply in this project due to the reasons explained in section 4.5 (p. 46); Also the pose determination that would be easy to obtain using these models will have to be carried out using a more classic approach.

The approach used takes advantage of SIFT, a classic computer vision algorithm. SIFT algorithm stands for Scale-invariant feature transform and is a feature detector proposed by David Lowe in 1999 [75] to detect and describe local features (keypoints) in images.

The main idea of our method is to create a database with several snapshots of various 3D models in different poses (angles of rotation) and then try to match the objects previously detected by the object detector with the snapshots in the 3D model database.

Firstly, it was necessary to create some 3D models that could represent the classes of objects detected by the object detector. Therefore, eight different 3D models were created using the Blender program [3]: a desk, a laptop, a computer monitor, a computer mouse, a keyboard, 2 types and chairs (office chair and bedroom chair), and a bed.

Secondly, a dataset was created with the photographs of the 3D models previously made in various positions. Eight photos were taken for each 3D model, corresponding to eight angles of rotation (0º, 45º, 90º, 135º, 180º, 225º, 270º, and 315º).

Thirdly, the SIFT algorithm [75] was applied to all images of the 3D model dataset created before, generating a database with the descriptors and local features for each 3D model snapshot in a given pose.

Subsequently, a Python program was created using the OpenCV library [21]. This program receives as input the panorama image in analysis and the coordinates of the object's bounding boxes obtained in the object detection step and outputs the selected 3D models to represent the objects in the scene and their respective poses. The main program steps are:

1. Computation of local features (keypoints) and descriptors for all the objects in the panorama image, using the SIFT algorithm [75].

2. Matching the descriptors of the objects' input image with the descriptors of the snapshots taken for each 3D model in different poses using the FLANN matcher [77].

3. Application of Lowe's ratio test [75] to eliminate bad matches. Following Lowe's reasoning, the match with the smallest distance is the best match for a given keypoint, and the match with the second-smallest distance is the equivalent of random noise. Therefore, if the best

[3] https://www.blender.org/

match can not be differentiated from the second-smallest match, then the best match should be rejected because it does not bring any information. So the principle is that there needs to be enough difference between the best and second-best matches. Since all the noise has been excluded, the number of matches between the images is significantly reduced, leaving only matches with information value, which improves the program's accuracy.

4. Selection of the best match between the input scene objects and the 3D models' snapshots, that is the one with more matches between keypoints. The pose is determined by the rotation angle of the best match view.

At the end of this phase, a 3D reconstruction of the input scene with all its principal objects represented by an equivalent 3D model, in the right pose and size is expected. Although, the reconstruction results may need some posterior manual adjustment.

## 4.7 Experimental Methodology

In the interest of validating whether or not the solution implemented achieves the desired objectives and answers the current issues, we will perform various experiments in each main task of the proposed pipeline. The results of each scenario will be analyzed critically and evaluated according to the standard metrics used in each task. Furthermore, we will compare our results with all the other state-of-the-art methods' results of that specific task whenever possible.

This evaluation will be presented throughout the chapter 5 (p. 51).

## 4.8 Summary

In this chapter, we started discussing the current issues of the analyzed approaches in chapter 3 (p. 25), finding a possible solution for them. Then, in section 4.2 (p. 42), having in mind the current state-of-the-art problems, we proposed a methodology to solve them, detailing all the steps briefly. Then, it is presented a sequence of four sections where is described the method used in the Layout Estimation task in section 4.3 (p. 44), in the Object Detection task in section 4.4 (p. 45), in the 3D Position Reasoning Task in section 4.5 (p. 46), and in the 3D Model Selection and Object's Pose Estimation Task in section 4.6 (p. 49). Lastly, the experimental methodology is detailed in section 4.7 (p. 50).

# Chapter 5

# Experimental Setup and Results

This chapter describes, in detail, the experimental setup, the evaluation, and the final results of all the steps needed to follow the proposed solution pipeline described in section 4.2 (p. 42). The next sections in this chapter correspond to the evaluation and discussion of an essential step in the proposed pipeline explained in section 4.2 (p. 42).

Section 5.1 (p. 52) explains the experimental setup details of the proposed solution's first step: **Layout Estimation**, describing realized experiments, the used dataset, and the quantitative and qualitative results. Section 5.2 (p. 58) similarly to the previous section, describes all the experimental setup details of the proposed solution's second step: **Object Detection**, describing realized experiments, the used dataset, and the quantitative and qualitative results. Section 5.3 (p. 65) analysis the results of the proposed solution's third step: **3D Position Reasoning**. Section 5.4 (p. 66) depicts the results of the last crucial step in the proposed pipeline: **3D Model Selection and Object's Pose Estimation**. Finally, section 5.5 (p. 71) summarizes this chapter.

## 5.1 Layout Estimation Task

To estimate the 3D room layout from a single panorama image, we employed the HorizonNet method [92] described in detail in section 4.3 (p. 44). To train, test and validate the model we used the Google Colab service [1].

### 5.1.1 Dataset

We train, evaluate and test the model using the dataset proposed in the LayoutNet work [109] . The dataset consists in the combination of the PanoContext dataset [105] with the extended Stanford 2D-3D dataset [18] annotated by Zou et al. [109].

The dataset contains 1062 equirectangular RGB panorama representing mostly large/medium scale environments, including offices, corridors, and open-spaces, like meeting rooms and auditoriums. A bar chart describes the diversity of scene types in figure 5.1.

By analyzing the bar chart in figure 5.1, we can conclude that 85% of the images in our dataset represent offices or open-spaces, and only 6% refers to corridors.

[1] https://colab.research.google.com/

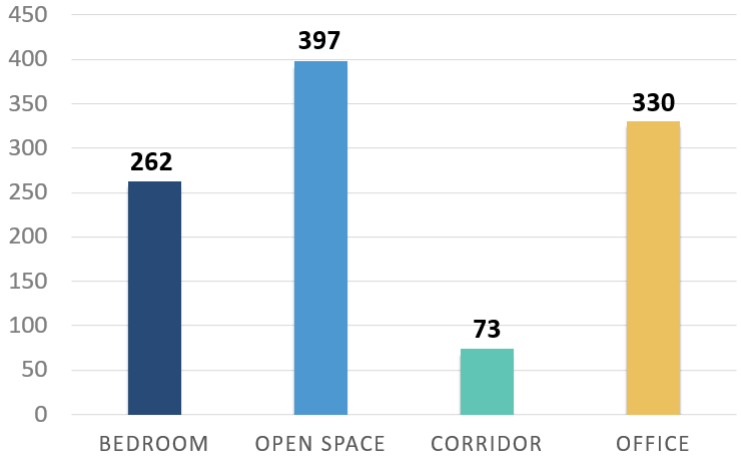



Figure 5.1: Distribution of scene types in the used dataset.

Of the 1062 images in the dataset, 65 of those have a general shape labeled by Cheng et al. [92], more complex than cuboid, like "L-shaped" rooms or "T-shaped" rooms. All the other images are annotated as a cuboid room. In figure 5.2, a bar chart represents the layout's complexity distribution of the used dataset.

By analyzing the bar chart in figure 5.2, we can assume that 86% of the panorama images in our dataset represent cuboid rooms, that is, with eight corners. So, only 14% of our dataset is composed of layouts more complex than cuboids.

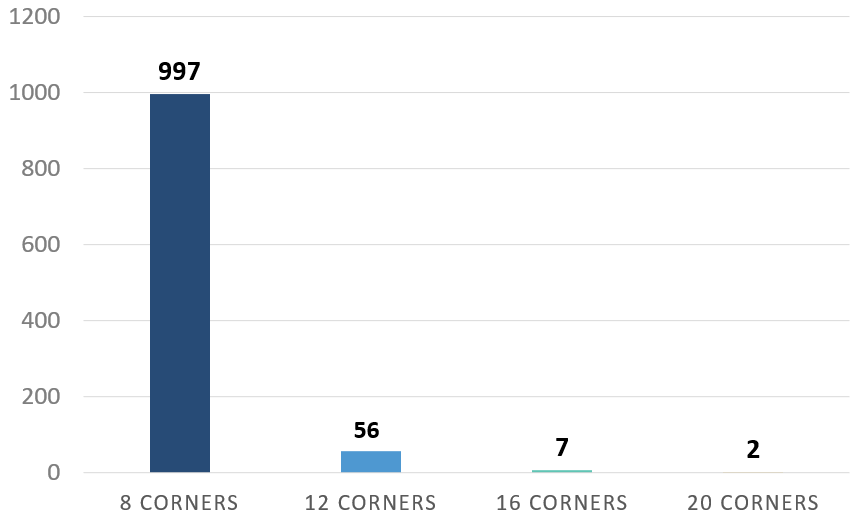



Figure 5.2: Distribution of the number of corner in the used dataset.

We adopt the same train/validation/test split of the one used for LayoutNet [109], with 78% of the images to train, 15% to test and 7% to validate the network. This distribution is represented in

the figure 5.3 bellow.

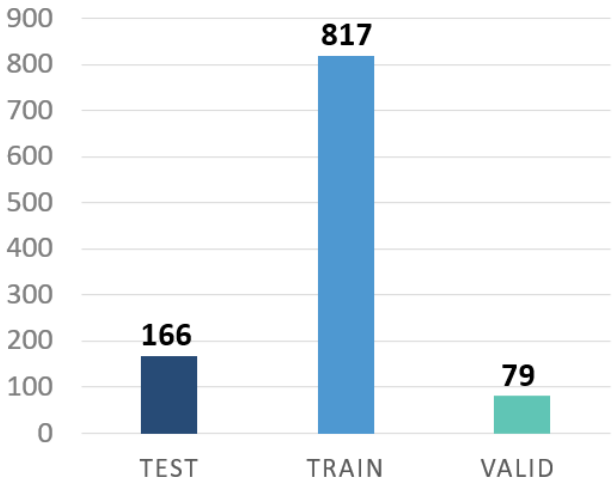



Figure 5.3: LayoutNet's [109] train/validation/test split.

### 5.1.2 Training Details

We decided to carry out two training experiments, where we used two different backbones, one deeper than the other.

In the first experiment, the Adam optimizer [57] was employed to train the network for 150 epochs with a batch size of 4 and a learning rate of 0.0001, using ResNet-50 [18] as the backbone.

In the second experiment, we used the same hyper-parameters as in the first one, except for the backbone. The ResNet-101 [47] was the adopted feature extractor.

As we can see in the graph represented in figure 5.4, as the number of epochs increase, as expected in both experiments with different backbones, the accuracy also increases, tending to settle close to the accuracy value of 80%.

We also can conclude that the experiment with ResNet-101 [47] as the backbone is the one with the highest accuracy, with a value of 79.25%. The difference between the accuracy obtained by the two convolutional neural networks is probably due to the larger size and capacity of ResNet-101 [47] compared to ResNet-50 [18].

In some deep convolutional neural networks, with the depth increasing, accuracy gets saturated, which leads to degradation. However, due to the use of residual networks, the ResNet networks can easily increase accuracy, maintaining the same hyperparameters and increasing the number of layers in the network substantially.

We also applied some data augmentation techniques in both experiences, including brightness change, panoramic horizontal rotation, and left-right flipping.

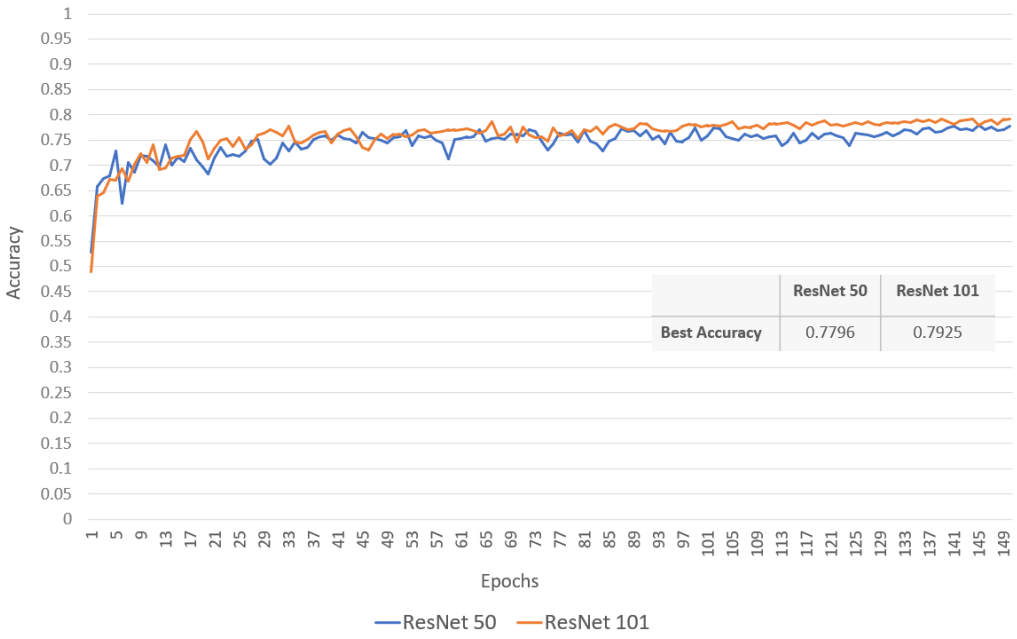



Figure 5.4: Accuracy evolution throughout training in both experiments: with ResNet-50 and ResNet-101.

### 5.1.3 Results

#### 5.1.3.1 Quantitative results

We evaluate our results on three standard metrics used in the layout estimation task: (a) 3D Intersection over Union (3D IoU), (b) Corner error, and (c) Pixel error. These metrics are explained in more detail in section 2.2.2 (p. 18). The corner error and pixel error are only calculated for predicted cuboid layouts.

Table 5.1 presents the two experiments' evaluation and the LayoutNet method results [109] in the dataset proposed by them, consisting of a combination of the PanoContext dataset [105] with the extended Stanford 2D-3D dataset [18].

Analyzing the table, we can conclude that both of ours networks have greater results than the LayoutNet model [109] in all metrics. Being the network with the ResNet-101 [47] as backbone, the one with slightly better results overall.

Table 5.1: Evaluation on the LayoutNet proposed dataset [109] . Bold numbers indicate the best performance.

| Method | 3D IoU (%) ↑ | Corner error (%) ↓ | Pixel error (%) ↓ |
|---|---|---|---|
| HorizonNet with ResNet-50 | 83.29 | **0.69** | 2.13 |
| HorizonNet with ResNet-101 | **84.03** | 0.70 | **2.11** |
| LayoutNet [109] | 75.12 | 1.02 | 3.18 |

Table 5.2 compares our quantitative results with other state-of-the-art method's quantitative results, evaluated on the Pano Context dataset [105]. As we can see, the HorizonNet model [92] outperforms the existing method under all metrics.

Table 5.2: Quantitative results of layout estimation evaluated on the PanoContext dataset [105].

| **Method** | **3D IoU (%)** ↑ | **Corner error (%)** ↓ | **Pixel error (%)** ↓ |
|---|---|---|---|
| PanoContext [105] | 67.23 | 1.60 | 4.55 |
| LayoutNet [109] | 74.48 | 1.06 | 3.34 |
| DuLa-Net [100] | 77.42 | - | - |
| CFL [37] | 78.79 | 0.79 | 2.49 |
| HorizonNet [92] | **82.17** | **0.76** | **2.20** |

#### 5.1.3.2 Qualitative results

To better visualize the results, the generated 1D representation is converted to a set of corner coordinates, as shown in the figure 5.5. Then, these coordinates are written in a JSON file format and so the 3D model can be generated using ThreeJS [2].

```
floor coord:[[-0.1959936   2.0545576 ]
 [-0.19599354  3.9839153 ]
 [-1.3569014   3.9839144 ]
 [-1.3569022  -4.093592  ]
 [ 2.9097128  -4.0935926 ]
 [ 2.9097135   1.1095303 ]
 [ 1.0311482   1.1095303 ]
 [ 1.0311477   2.0545578 ]]
```

Figure 5.5: Example of the layout estimation module's output file.

When there are walls with different heights, that is, some unevenness between the rooms, it appears that in most cases, the model cannot effectively detect the corners. In addition, sometimes, when the room is very large, and there is low contrast, the geometry of the most distant walls, if they have complex shapes, can be simplified to a single wall giving rise to a cuboid layout. Figure 5.6 presents some qualitative results that illustrate the problems mentioned earlier. The figure shows the input panorama image, the 1D representation in the green line, and the problematic area flagged with a red rectangle.

One of the possible main reasons for these is that the training dataset is quite unbalanced, with a strong predominance of cuboid-shaped rooms. Furthermore, the steep unevenness of the corner's height position and the reduced contrast and lightness should confuse the model.

[2] https://threejs.org/

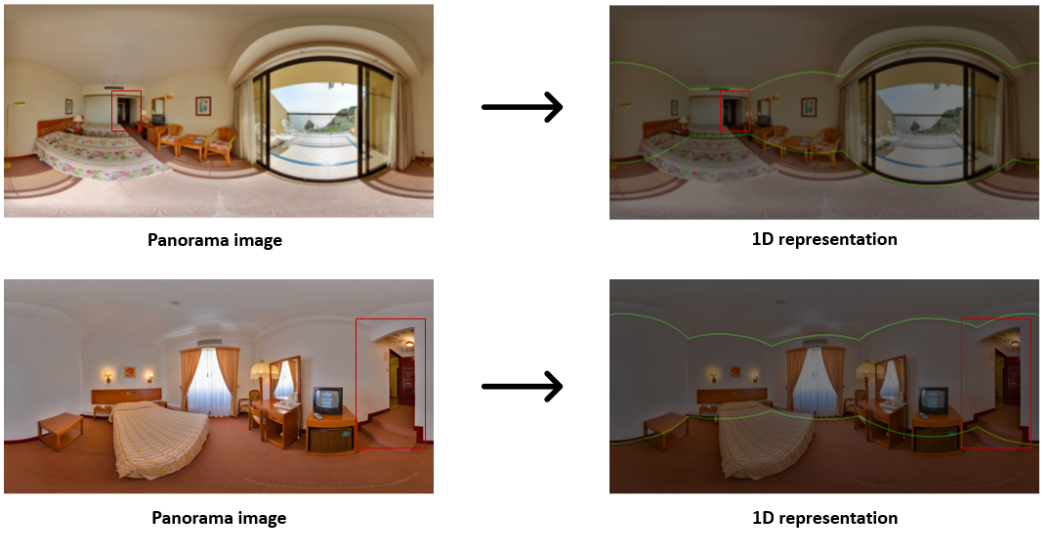



Figure 5.6: Qualitative results of layout estimation phase that illustrate some of the model's weaknesses. The red rectangles symbolize the problematic areas. These limited areas indicate the unevenness between the rooms that the model could not detect, so a more generic cuboid layout was estimated (green line).

Figure 5.7 shows some of the successful layout estimation's qualitative results, presenting the input image, the 1D representation in the green line, and the final 3D layout visualized in ThreeJS[3].

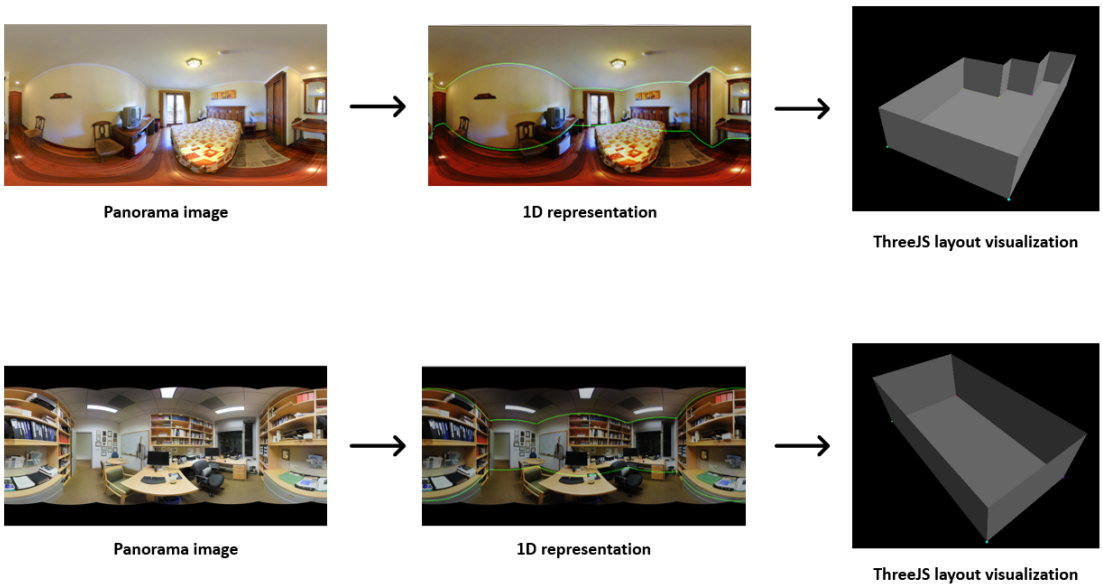



Figure 5.7: Successful qualitative results of layout estimation phase.

[3]https://threejs.org/

## 5.2 Object Detection Task

To detect the main objects present in the panoramic image, we use the YOLOv4 method [19], which is explained in detail in section 4.4 (p. 45). To train, test, and validate the model, as well as in the previous phase, we take advantage of the Google Colab service [4].

### 5.2.1 Dataset

We built a dataset composed of perpective indoor images extracted from Open Images [60]. and some panorama indoor images selected from the combination of the PanoContext dataset [105] with the extended Stanford 2D-3D dataset [18]. During the data collection process, we selected a set of indoor environments: bedrooms, offices, and open-spaces, and a group of objects that are usually found in each of these divisions: bed, laptop, computer monitor, computer mouse, keyboard, chair, and desk/table.

To extract images that possess these objects and the respective annotations from the Open Images Dataset [60], we used OIDv4 ToolKit[5], a tool developed to extract images that belong to specific classes of objects from the Open Images dataset [60].

After obtaining the images, we went through a manual process of validation and creation of the images' labels. To correct some badly labeled annotations and add some missing labels, we used Roboflow [6], a tool that provides an easier way to organize and annotate image data.

After collecting, cleaning, and labeling the data, we analyzed it to verify whether it is balanced and to visualize other characteristics. The dataset is composed of 991 images, and on average, each image has four annotations.

Regarding object distribution in the pictures, the dataset is not well balanced, as it contains significantly fewer beds than other objects, as we can see in Figure 5.8,

[4] https://colab.research.google.com/
[5] https://github.com/EscVM/OIDv4_ToolKit
[6] https://roboflow.com

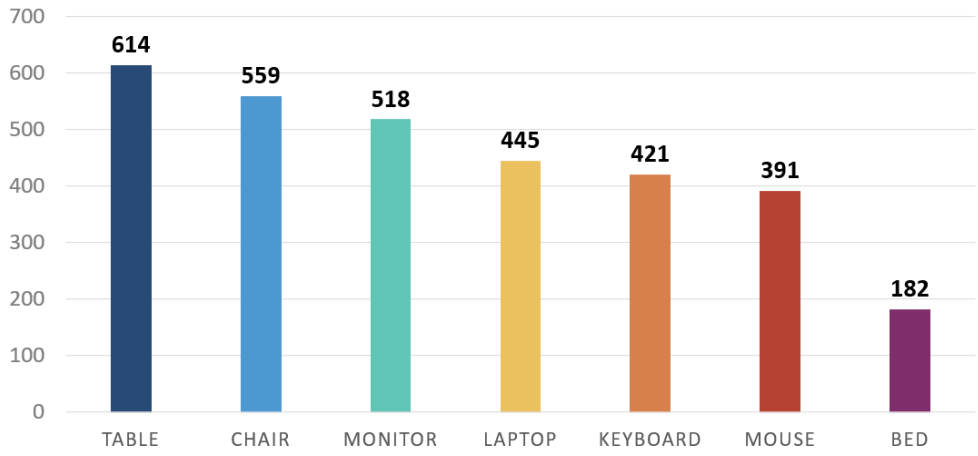



Figure 5.8: Object distribution.

We split the data in 70% for training, 20% for validation and 10% for testing, giving rise to the distribution that is represented in figure 5.9.

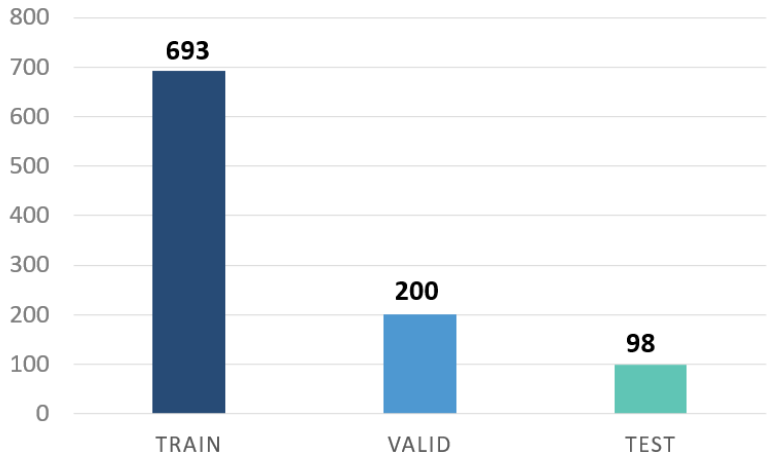



Figure 5.9: Used train/validation/test split.

Concerning the image type, our dataset is composed by 90% of perpective images and 10% of panorama images. The inclusion of panoramic images in the dataset is because our ultimate goal is to detect the maximum objects in a panorama image given as input to reconstruct the 3D scene model. So, to increase the model's performance in this type of images, we include some panorama

images in the dataset, and thus the model can learn under these images' characteristic distortions. The distribution of images' types in the dataset is represented in the figure 5.10.

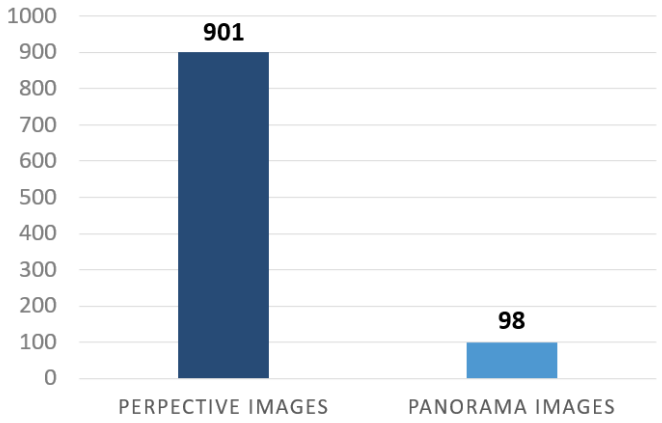



Figure 5.10: Distribution of images' type in the dataset.

### 5.2.2 Training Details

In the object detection phase, before starting the model training, we decided to resize all images to a height and width of 416 to reduce the computational time during training. Besides, we selected a saturation and exposure of 1.5 and a hue of 0.1 for the dataset.

Regarding the deep neural network hyper-parameters, we chose a batch size of 64 images, and each batch was split into 16 subdivisions. The dataset was shown 1600 times to the network (1600 epochs). We chose a learning rate of 0.001, a momentum of 0.949, and a decay of 0.0005.

In this experience, we used the Complete Intersection over Union (CIoU), explained in detail in section 4.2.2, as loss function since it is the one that achieves the best results according to Bochkovskiy et al. [19].

The chart represented in figure 5.11 presents the evolution of the mean average precision (mAP) and the loss function (CIoU) through the epochs of training. In the chart, the red line represents the mean average precision, and a blue line marks the loss function values.

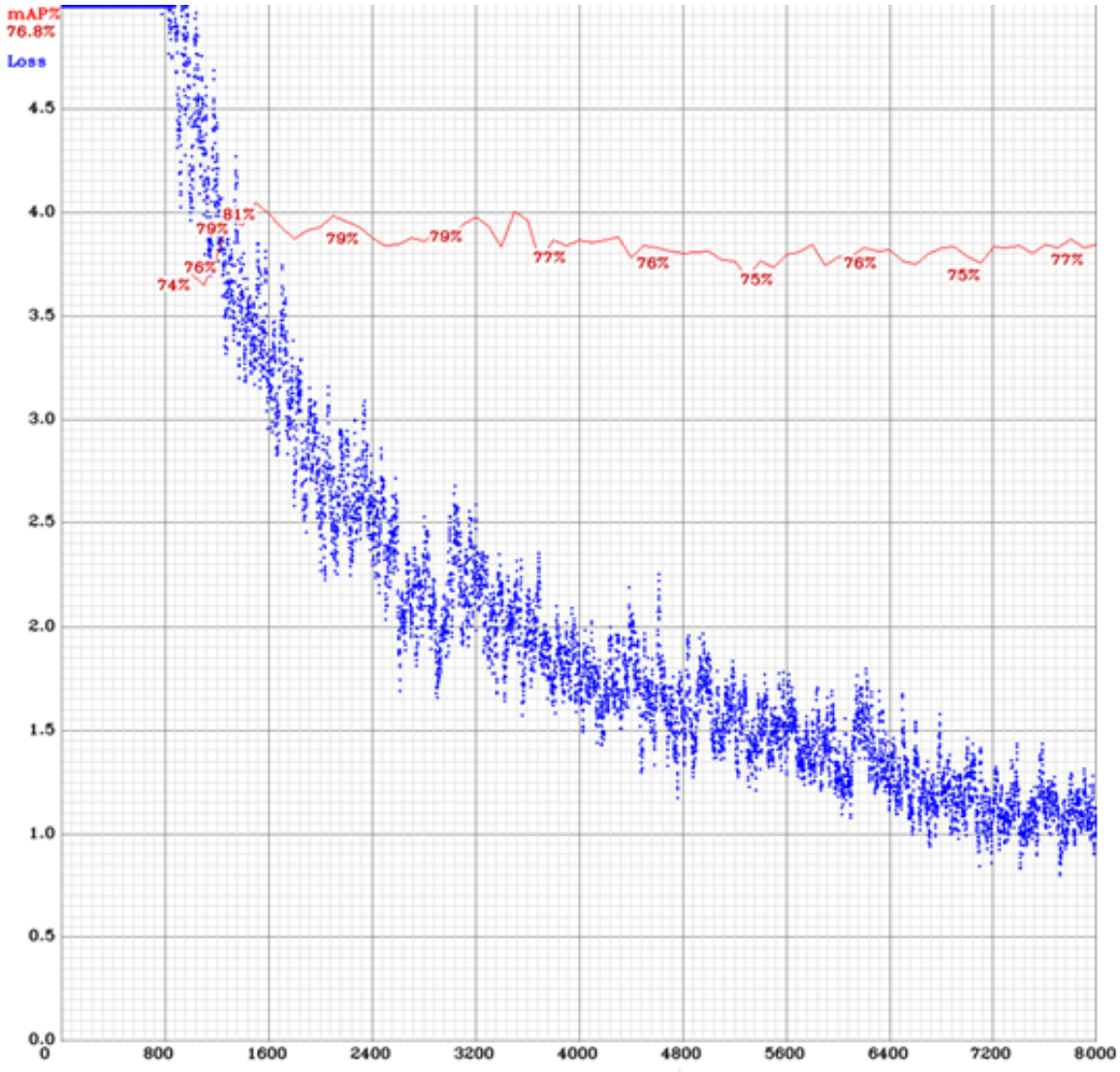



Figure 5.11: Mean average precision (mAP) and loss function evolution through training.

As we can see, in the last 1000 iterations, the value of the loss function tends to stabilize at around 1. This fact indicates that the network is reliable, and so, we can stop training. Furthermore, the mean average precision (mAP) is also stable and does not appear to increase in the future.

### 5.2.3 Results

#### 5.2.3.1 Quantitative results

We evaluate our results on two standard metrics used in the object detection task: (a) Mean average precision (mAP) and (b) Intersection over Union (IoU). These metrics are explained in more detail in section 2.2.3 (p. 19).

Table 5.3 contains an overview of the quantitative results of YOLOv4 in our dataset. We obtained an average intersection over union (IoU) of around 66%, which is fairly good since the

predicted bounding boxes overlap with more than 66% of the true bounding boxes, but it has room to improve.

Regarding the F1 score, the result is considered reasonably good since it is around 80%, which means that we have low false positives and false negatives, and so, the model is correctly identifying real threats.

The precision and recall results are also good, with values of 82% and 80%, respectively. The high precision value means that the number of false positives is low and the number of true positives is high, and so the model is assigning the class correctly to the detected objects. The high value of recall indicates that the number of true positives is very close to the total number of actual relevant objects, and so the model detects almost all objects in the image.

However, the mean average precision (mAP) results are not so satisfactory, with a value around 77%.

Table 5.3: Quantitative results of object detection evaluated on the dataset described above.

| **Metric** | **Obtained value (%)** |
|---|---|
| Average intersection over union (IoU) | 66.43 |
| F1 measure | 80.92 |
| Precision | 82.3 |
| Recall | 79.6 |
| Mean average precision (mAP@0.50) | 76.86 |

The average precision results obtained for each object can be seen in figure 5.12.

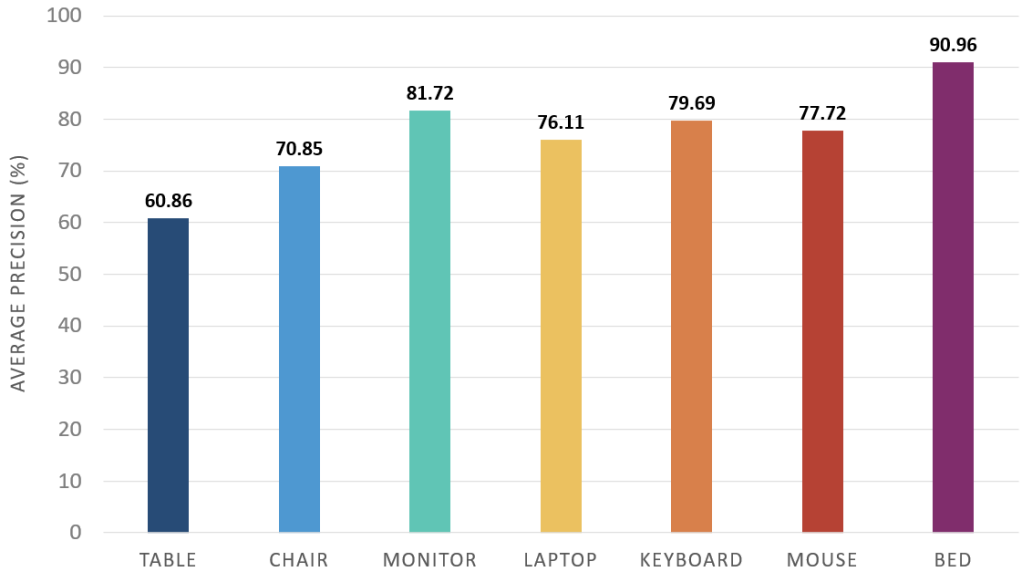



Figure 5.12: Results of average precision (AP) for each object.

In the bar chart of figure 5.12, we can see that table and chair, although they are the most represented objects in the dataset, have the lowest average precision (AP). In contrast, the bed, the less represented object in the dataset, is the one with the highest average precision (AP). This is mainly due to the number of occlusions that occur in type chair and table objects compared to objects of type bed.

Usually, tables and chairs are always heavily occluded, in the case of tables by the objects on top of them, such as laptops, monitors, mouses, and keyboards, and in the case of chairs by the tables themselves or by people who are using them.

Excluding tables and chairs that have a lower average precision, the results of average precision are reasonably good, varying between 76.11% and 90.96%.

#### 5.2.3.2 Qualitative results

Although the model has a good mean average precision (mAP) and, therefore, good results, sometimes it fails. Most of the time, these errors are omissions, and the wrong classification of an object is extremely rare.

The main reasons for the non-detection of particular objects are highly connected with the object's size and their respective distance to the camera. The smaller the object and the furthest from the camera an object is, the more difficult it will be to detect. The low contrast, the low light as well as the object's occlusions also jeopardize the detection.

One of the possible solutions for the issues stated above is to increase the images' resolution in the dataset. The problem with this solution is that it would make training time too long and unreasonable for the available resources.

Figure 5.13 shows some examples of misdetection and detection omission previously discussed.

Figure 5.14 presents some of the successful object detection's qualitative results. In each image, we can see the different object's bounding boxes in various colors with the respective labels.


Figure 5.13: Qualitative results of object detection phase that illustrate some of the model's weaknesses.


Figure 5.14: Successful qualitative results of object detection phase.

The object detection output file is in the form of a detected objects list. Each object has associated the respective class and an array that contains the detection confidence value, the left x and top y pixel coordinates of the bounding box, and the bounding box's width and height.

Figure 5.15 illustrates an output example of the object detection phase.

```
objects:
Table: 99% (left_x:   410   top_y:  106   width:  120  height:   30)
Computer-monitor: 100% (left_x:  35   top_y:  108   width:   35   height:   35)
Computer-monitor: 87% (left_x:  10   top_y:  109   width:   43   height:   43)
Computer-keyboard: 100% (left_x:  60   top_y:  108   width:   32   height:   15)
Chair: 100% (left_x:  80   top_y:  117   width:   70   height:   45)
Chair: 8% (left_x:  155   top_y:  120   width:   30   height:   17)
Computer-monitor: 93% (left_x:  200   top_y:  108   width:   24   height:   34)
Chair: 96% (left_x:  210   top_y:  120   width:   28   height:   39)
Table: 75% (left_x:   226   top_y:  112   width:  135  height:   40)
Table: 84% (left_x:   254   top_y:  134   width:  124  height:   50)
Computer-monitor: 96% (left_x:  295   top_y:  109   width:   52   height:   43)
Computer-keyboard: 100% (left_x:  290   top_y:  112   width: 47  height:   17)
```

Figure 5.15: Example of the object detection module's output file.

## 5.3 3D Position Reasoning Task

The main objective of this task is to estimate as accurately as possible the 3D location of the objects previously detected in the preceding phase and thus position them within the estimated room's geometry obtained in the layout estimation phase. The method used in this task is explained in detail in section 4.5 (p. 46).

### 5.3.1 Results

Due to the lack of ground truth, it is impossible to make a rigorous evaluation of the results. However, through the analysis of the images, we can perceive the quality of the results.

Figure 5.16 depicts some results of the 3D position reasoning step. The figure shows the input panorama image with the bounding boxes of the detected objects in the previous phase and the corresponding 3D scene represented in ThreeJs[7].. The color of the bounding boxes characterizes each type of object, and the same happens in the 3D scene, that is, the correspondence between the objects' type and colors is preserved.

It should be noted that only objects detected with a confidence greater than or equal to 20% are represented. Hence the false positives that we see in the images in figure 5.16 do not appear in the 3D representation.

By observing figure 5.17, we can conclude that the results are very good, seeming very accurate both in the location of the objects within the previously estimated layout and in the estimation of the objects' size.

[7]https://threejs.org/

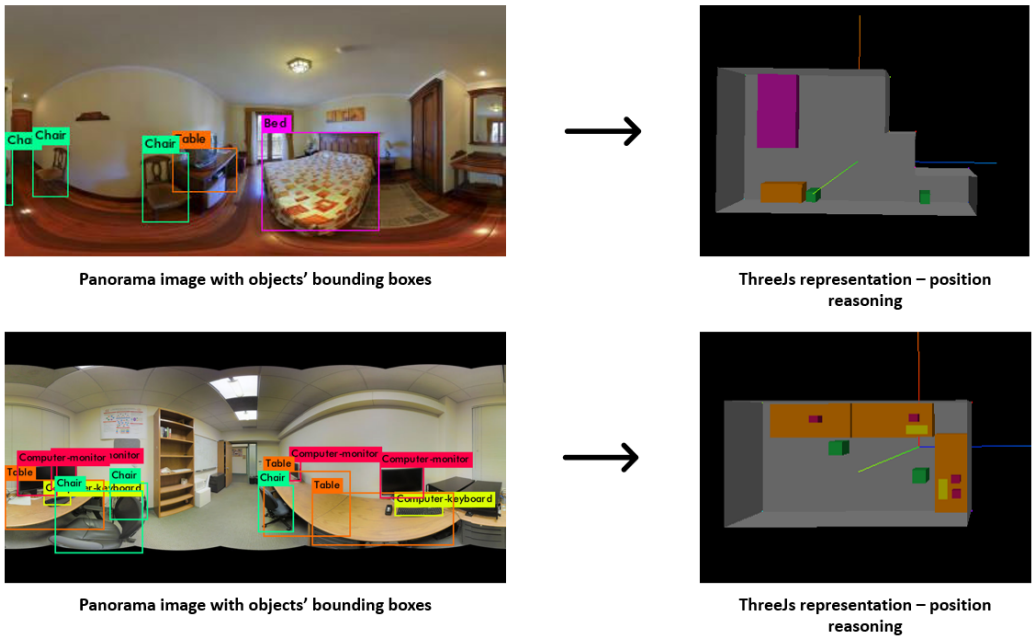



Figure 5.16: Results of 3D position reasoning step.

## 5.4 3D Model Selection and Object's Pose Estimation Task

At this stage, we already have the room's layout, all the main objects of the scene, their sizes, and their respective locations within the estimated geometry.

To complete the reconstruction of the scene in 3D to be carried out successfully, it is then necessary to do the automatic selection of 3D models to represent the objects and the choice of their pose. The object's pose refers to the angle that it makes relative to the camera. The method used in this task is explained in detail in section 4.6 (p. 49)

### 5.4.1 Results

#### 5.4.1.1 3D Model selection results

Regarding the selection of a 3D model, our results were excellent. In a set of 50 objects, which included examples from all classes, almost all the 3D models selected by the program correspond to the appropriate 3D models to represent the concerned object.

Figure 5.17 presents a chart showing the correct and wrong 3D model's selection for each object's classes.

As we can see by analyzing the graph, only one laptop has the wrong 3D model selected. In this case, the 3D model selected was a keyboard. Because the laptop in question was occluded, and all laptops are composed of a keyboard, we consider the error reasonable.

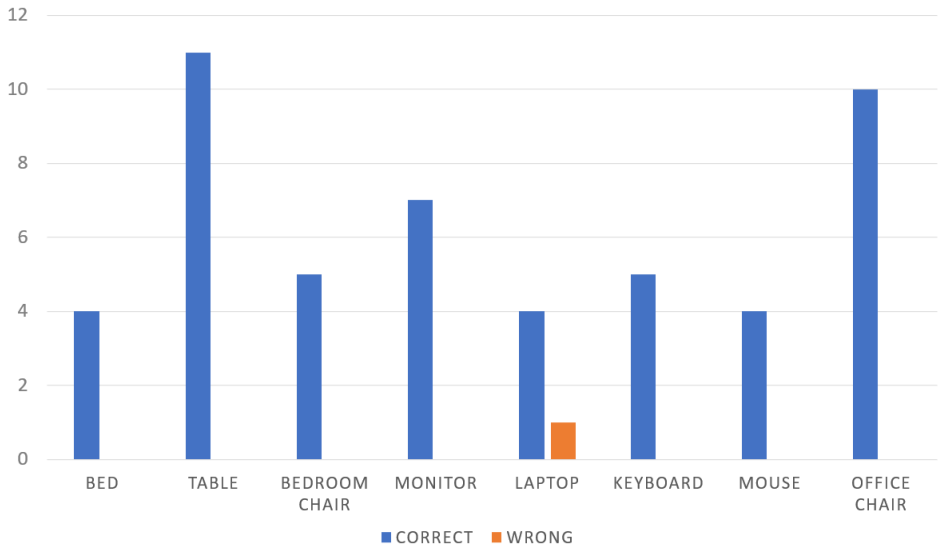



Figure 5.17: 3D model selection results.

### 5.4.1.2 Pose estimation results

Concerning the pose estimation, the results were not so satisfactory. Figure 5.18 presents a bar chart where we can perceive the number of correct and wrong pose estimations for each type of object. Above each bar, the percentage of correct and wrong pose estimations of each kind of object are represented.

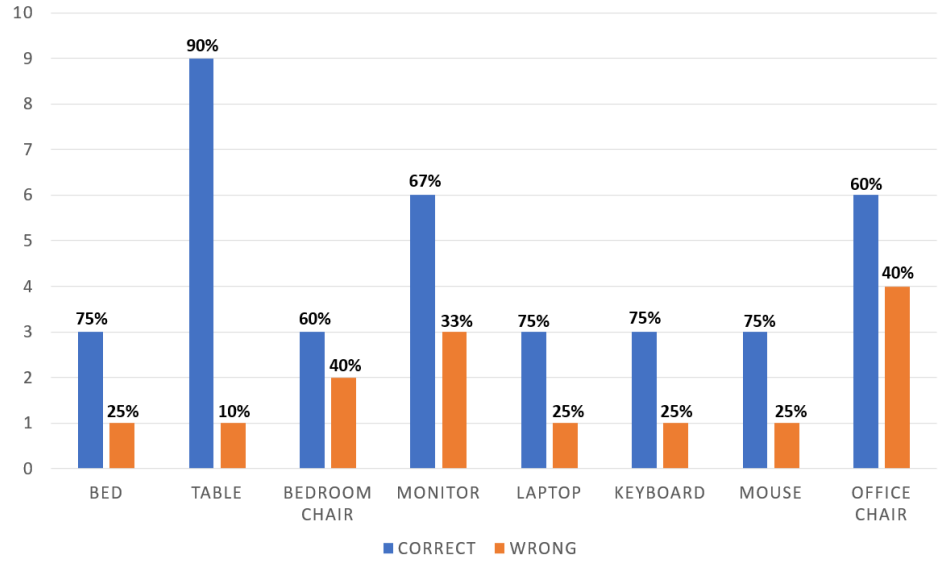



Figure 5.18: Results of pose estimation for each type of object.

Analyzing the bar chart in figure 5.18, we can conclude that the program is better at estimating the pose of some types of objects. Being excellent at estimating tables' pose (90% of the tables' poses were well estimated), much worse at estimating both the pose of bedroom chairs and office chairs and relatively good at estimating the rest object's types.

One of the reasons for this differences lies in the fact that some objects, namely tables, have fewer possibilities for pose because the 3D model that represents them is the same in the pose (0º and 180º), (45º and 225º), (90º and 270º), and in (135º and 315º), as we can see in figure 5.19.

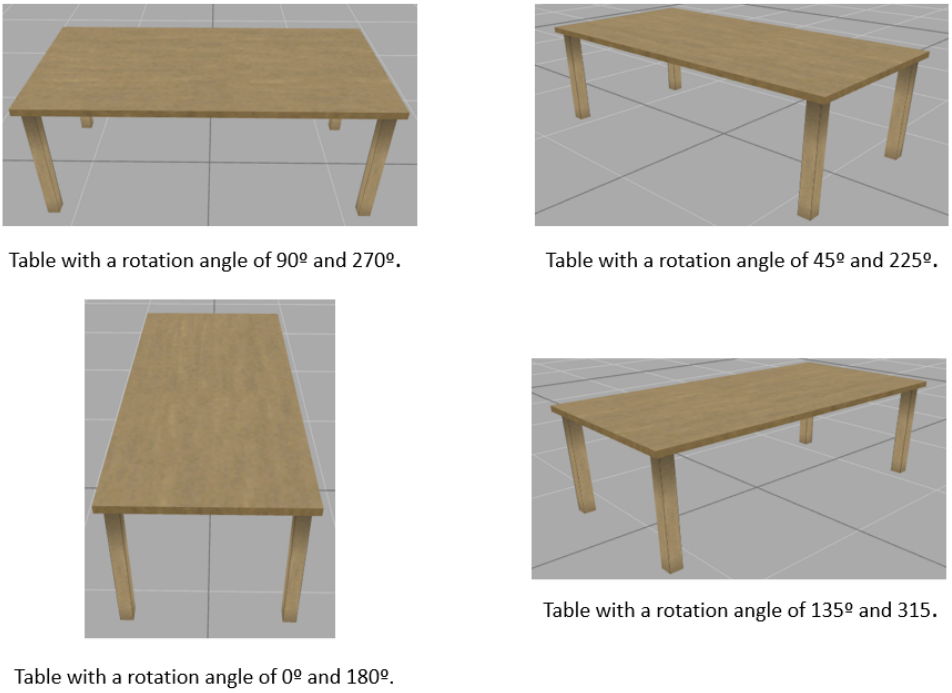



Figure 5.19: Table's poses. Due to object's symmetries, the table looks like it is in the same pose in the rotation angles 90º and 270º, 45º and 225º, 0º and 180º, and in 135º and 315º.

Figure 5.20 represents a bar chart where we can perceive the number of correct and wrong pose estimations for each pose angle. In the same way as the previous chart, above each bar is represented the percentage of correct and wrong pose estimations of each pose angle, respectively.

As we can see by analyzing the chart in figure 5.20, in all classes (angles), the percentage of correct predictions is never lower than the percentage of wrong predictions. On average, the rate of correct answers is 72.25%, which is a very acceptable result.

However, in some rotation angles, the program has a much better prediction rate. This substantial variance in the prediction rates is closely related to the characteristics of the SIFT algorithm [75] used to extract feature points and descriptors from images.

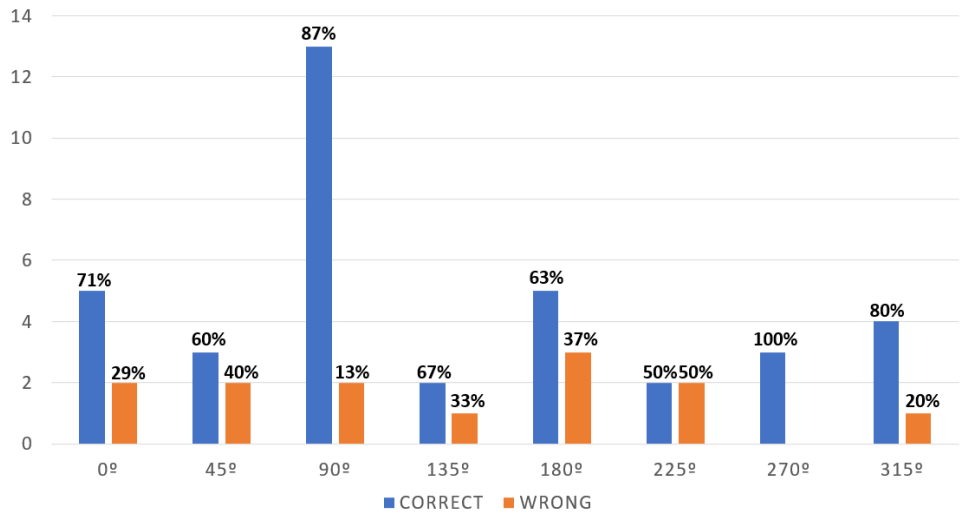



Figure 5.20: Results of pose estimation for each pose angle.

The SIFT algorithm [75] is invariant to image rotation and scale, this means that, in theory, the algorithm is not affected by image rotations or changes in size, identifying the feature points as the same. Because of this, the 90º and 270º angles have the highest prediction rate because, in most objects, symmetry does not occur in these angles. Thanks to these SIFT algorithm's characteristics and due to the strong symmetries that usually happen in objects, there is a great probability of confusing the poses (0º and 180º), (45º and 225º), (90º and 270º), and in (135º and 315º).

Figure 5.21 shows an example of the relationship between the angle of rotation and the object's pose, in this case an office chair.

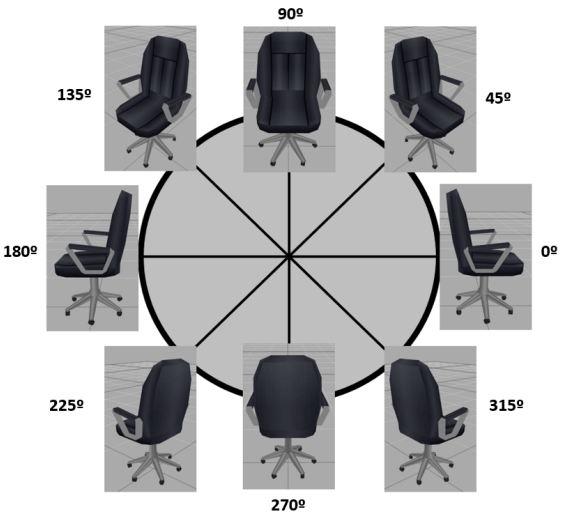



Figure 5.21: Example of the relation between the rotation angle and the object's pose.

#### 5.4.1.3 Final results

At the end of this phase, all the main steps in our pipeline are completed, and so it is possible to do a 3D scene reconstruction from only one panorama image as input.

In order to make the scene more realistic, we decide to either apply the predominant color from both floor and walls of the input panorama image in the walls and floor of the generated 3D scene or apply the panorama image itself as a texture in generated 3D scene's walls and floor.

To extract the predominant color of the walls and floor, first, the panorama image has been cut into two parts, one representing the floor and the other representing the walls. Second, a clustering algorithm, in this case, the k-means algorithm [74] with 4 clusters, is applied to the two image parts. And so, each pixel of the image is assigned to one of the clusters according to its color. Lastly, the selected color as predominant corresponds to the biggest cluster.

Figure 5.22 presents an example of the final 3D scene with the predominant color applied to the walls and floor, respectively.

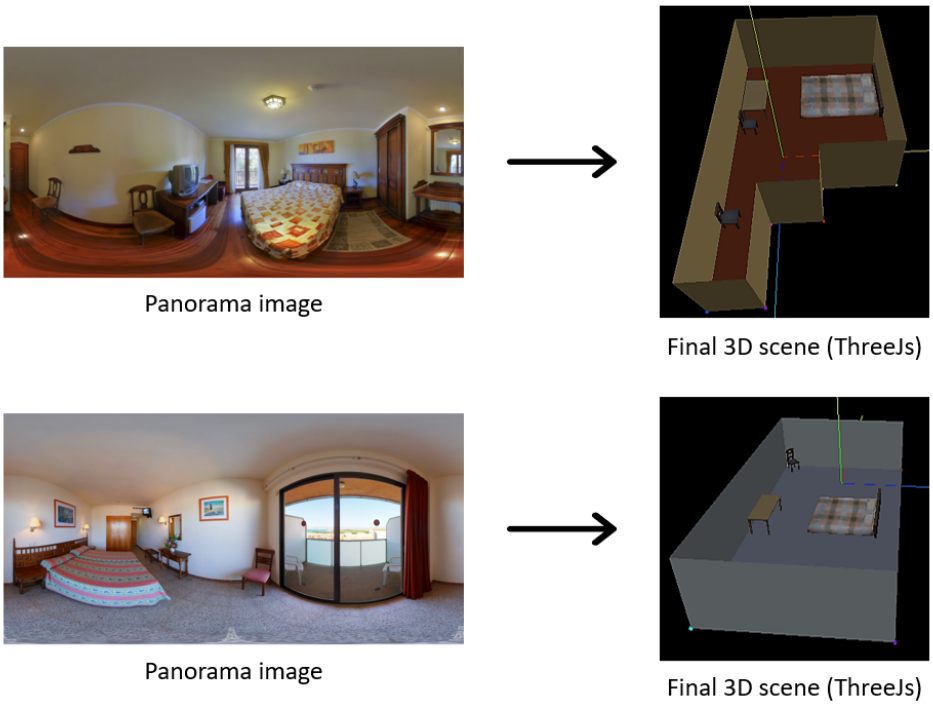



Figure 5.22: Final 3D scene with previous extracted predominant color in the walls and floor.

To apply the panorama image as a texture in the generated 3D scene's walls and floor, we used the correspondence between corners in the panoramic input image and the corners in the 3D model acquired in the estimation layout phase and converted it to texture coordinates.

Figure 5.23 shows us some examples of the 3D scene reconstruction.

As we can see, all the final 3D scene examples present a very accurate layout (walls and floor) with all the main scene objects within the room´s geometry. Furthermore, the objects' size, pose, and location are very similar to the reality shown in the input image. Also, the 3D model was very well automatically chosen to represent each of the detected objects. In sum, we reckon that the final results show a very realistic and faithful reconstruction of the 3D scene.

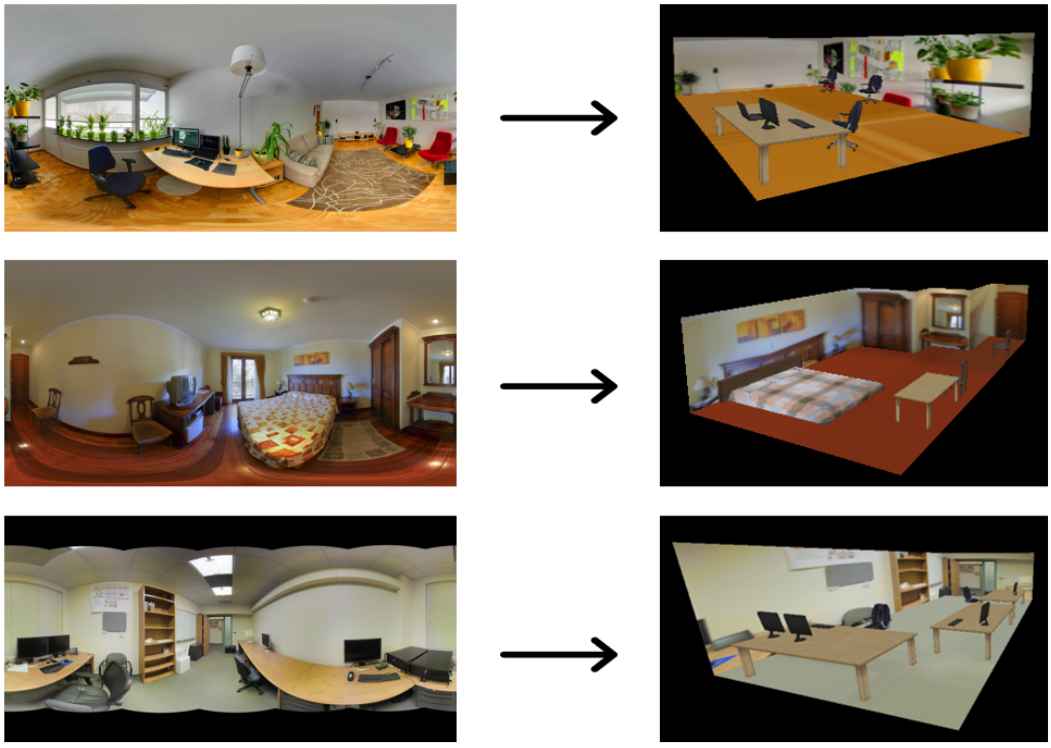



Figure 5.23: Some examples of 3D scene reconstruction from only one panorama image.

## 5.5 Summary

This chapter presented all the steps in the development process meticulously, evaluating the methods used and analyzing the results critically in all phases.

This chapter includes a sequence of four sections where is described the experimental setup, and the results of the **Layout Estimation task** in section 5.1 (p. 52), of the **Object Detection task** in section 5.2 (p. 58), of the **3D Position Reasoning task** in section 5.3 (p. 65), and of the **3D Model Selection and Object's Pose Estimation task** in section 5.4 (p. 66).

Throughout the chapter, we can conclude that all the steps in our approach had excellent results, consistently achieving satisfactory performance, thus employing a better solution than alternative approaches. Our results in the layout estimation and object detection tasks have surpassed

all the results obtained by previous methods. In the 3D position reasoning, 3D model selection, and object's pose estimation tasks, we proposed novel methods that proved extremely valuable in the 3D reconstruction of the scene presented in a panorama image and achieved very good results.

Although there is room for improvement in some of the steps, we concluded that our approach as a whole has a very satisfactory performance and achieves outstanding results, managing to fulfill all the projected goals and responding to all needs. Furthermore, our approach is the first one, to our knowledge, that from a single panoramic image builds a complete 3D scene, including the geometry of the room and its main objects represented by a 3D model chosen automatically, in the respective pose, size, and correct location.

# Chapter 6

# Application and Solution's Integration



This chapter describes the web/desktop application developed to visualize the 3D scene generated by the pipeline described in the previous chapters and explains the integration of our approach in the Critical Manufacturing MES. Section 6.1 (p. 73) is responsible for the **Web/Desktop application** development. It states the technologies used, and describes the main features. Section 6.2 (p. 75) describes the **integration of the solution in the FabLive 3D** component of the Critical Manufacturing MES. Finally, section 6.3 (p. 77) summarizes this chapter.

## 6.1 Web/Desktop application

A web application and a posterior cross-platform desktop application were developed with the primary objective of enabling the visualization of 3D scenes and allowing the adjustment or correction of some details in the scene, such as the object's rotation angle.

### 6.1.1 Technologies

The web application was developed in JavaScript, HTML, and CSS using a JavaScript library named ThreeJs [1]. Later, to simplify the use of the application for the end-user, we created a cross-platform desktop application using a framework called Electron[2].

### 6.1.2 Input and Output

The web/Desktop application receives as input the JSON file generated in the last step of the pipeline explained in detail in the previous sections. This JSON file includes all the necessary data to create a 3D scene automatically. It contains the information obtained by the layout estimation, object detection, 3D position reasoning, 3D model selection, and pose estimations tasks. An example of the input file can be consulted in Appendix A and is structured as follows:

- **"img path":** The panorama image name.
- **"dimensions":** Dictionary with the maximum coordinates of the room's layout in each dimension *(x, y, z)*.
- **"floor color":** The hexadecimal value of the floor's predominant color.
- **"walls color":** The hexadecimal value of the walls' predominant color.
- **"walls":** Array of dictionary elements, each element represent one of walls, and is structured as follows:
  - **"from":** The first pair of coordinates *(x1,z1)*.
  - **"to":** The second pair of coordinates *(x2,z2)*.
  - **"height":** Height of the wall.
- **"floor":** Array of floor coordinates pairs.
- **"walls texture UV":** Array of walls UV coordinates. It is used to apply the panorama image in the 3D walls as texture.
- **"objects":** array of dictionary elements, each element represent one of the detected objects, and is structured as follows:
  - **"id":** Type of object, states the 3D model selected to represent this object.
  - **"color":** Color of the object, each color is associated with one class of objects. If is not assigned a 3D model to represent the object, it will be represented in the 3D scene as a cube/parallelepiped with the respective color of its class.
  - **"position":** Dictionary with the localization coordinates of the object *(x,y,z)*.

[1] https://threejs.org/
[2] https://www.electronjs.org/

- **"size":** Dictionary with the size of the object *(x,y,z)*.
- **"rotation":** Angle of the object's rotation (pose).

The web/Desktop application allows exporting the scene in a JSON format that any ThreeJS [3] application can interpret. This JSON file includes all the information about geometries, materials, textures, images, shapes, and objects.

### 6.1.3 Main Features

Our application has the primary goal of allowing the 3D scene visualization and a second goal of enabling the end-user to correct and alter some details in the scene. For that being possible, we developed several features, listed below.

- Import of JSON files and creation of a 3D scene based on that information.
- Export of scenes in a JSON file format that any ThreeJS technology can understand.
- Option of adding the panorama image as a texture in the walls or paint it with the prominent color in the respective panorama image walls and floor.
- Option of changing the individual object's position, size, and rotation angle.
- Option of changing the all the same type object's size.

Figure 6.1 presents an example of a scene generated in our application.

## 6.2 FabLive 3D integration

FabLive 3D is a component in the Critical Manufacturing MES, which provides the visualization of the shop floor to its customers, allowing, for instance, to see the processes that are happening in real-time on each machine on the shop floor.

This component allows creating scenes manually, creating objects, or importing models for later manual positioning. In order to be able to integrate our solution in this environment in the least intrusive way possible, a new feature has been added to FabLive.

This new feature enables the import of complex scenes in a JSON format. In this way, it is possible to view and, if necessary, modify the 3D scene automatically created from a panoramic photograph in the Critical Manufacturing MES environment.

Figure 6.2 shows an example of a scene generated by our approach imported into the FabLive 3D component.

[3] https://threejs.org/

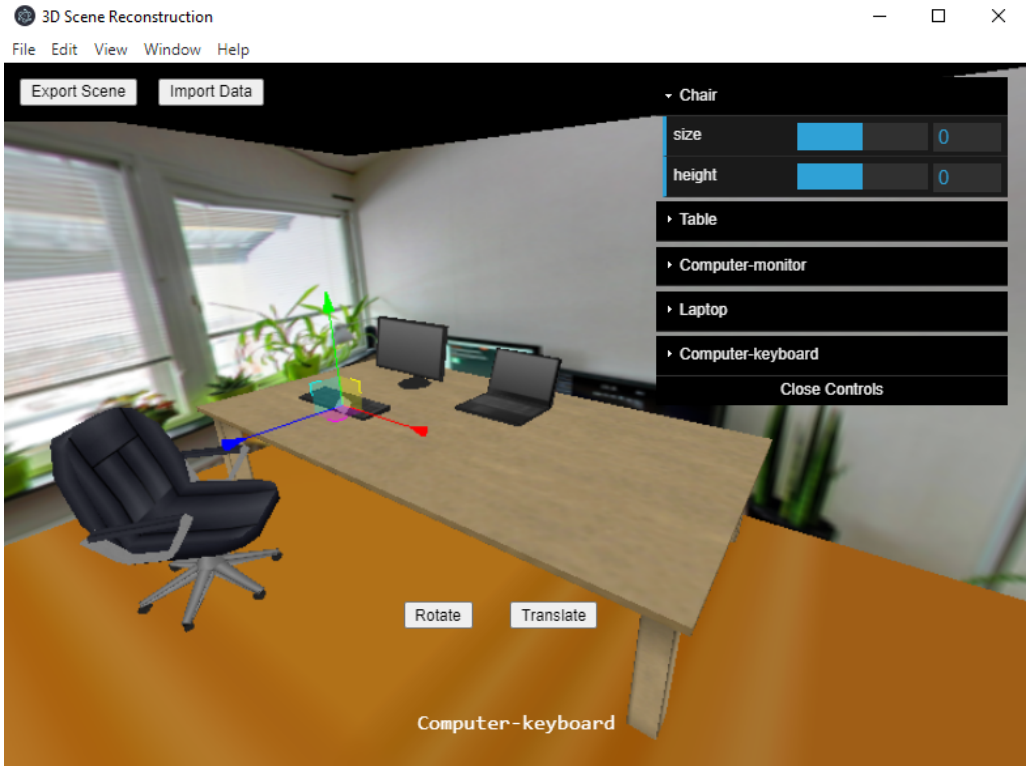



Figure 6.1: Screenshot of our application.

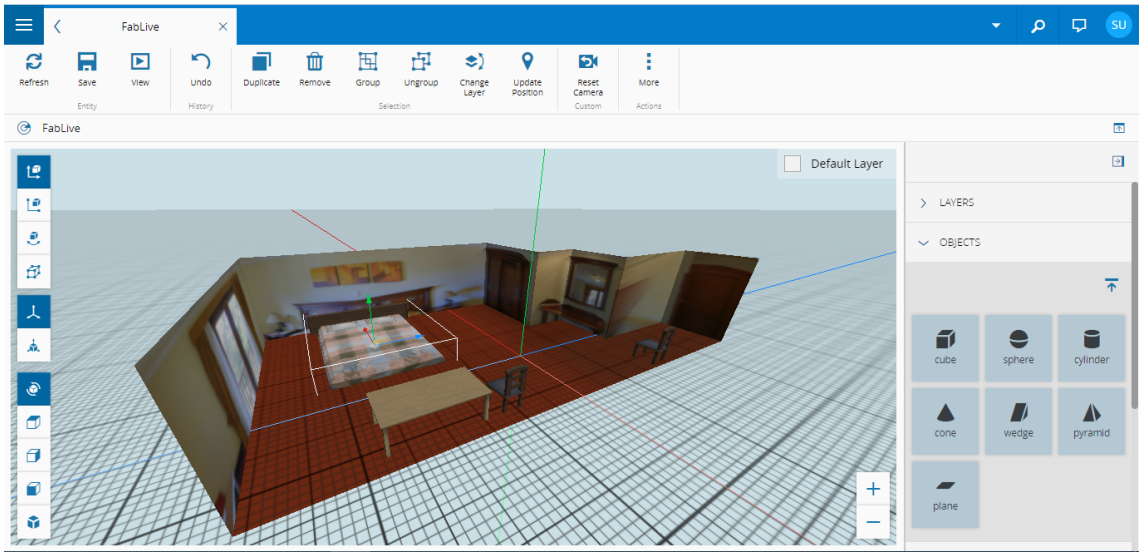



Figure 6.2: Screenshot of FabLive 3D component showing a imported scene.

## 6.3 Summary

In this chapter, we presented the developed web/desktop application. The first section described the application's input and output, its main features, and the technologies used to develop the application. Thus, we can conclude that this tool is essential and indispensable for achieving our dissertation's goals since it allows the visualization and modification of the generated 3D scene.

In the second section, we explain how our solution was integrated into the Critical Manufacturing MES. In summary, to incorporate our solution in the least intrusive way possible, we added a new feature to the FabLive component. This new feature enables the import of complex scenes in a JSON format and thus allows the visualization and modification of the generated 3D scene in the Critical Manufacturing MES environment.

# Chapter 7

# Conclusions and Future Work



This chapter presents a summarization of this dissertation. Section 7.1 (p. 79) presents an overview of the developed work and exposes the ultimate conclusions of this project. Section 7.2 (p. 80) outlines the contributions made during this dissertation. Section 7.3 (p. 80) describes the difficulties faced during the development of the solution. Finally, section 7.4 (p. 81) lists opportunities not explored during the development phase and improvements or enhancements to our final solution.

## 7.1 Conclusions

Throughout the development of this dissertation, we faced numerous challenges that not only made us think how powerful deep learning is but also that there are innumerable things to be explored and studied in the 3D scene understanding area.

The area of 3D scene understanding, namely 3D scene reconstruction from only one photograph, is an extremely recent field that is becoming very popular because of its numerous applications. Most state-of-the-art methods focus on just one of the 3D scene reconstruction sub-tasks, such as object detection or layout estimation, and very few tried to develop a pipeline to reconstruct a 3D scene based on images.

Our pipeline presents a novel complete approach to do the 3D scene reconstruction from only one panorama image. Our method answers all the necessary sub-tasks: layout estimation, object detection, 3D position reasoning, pose estimation, and 3D model selection to achieve an accurate 3D scene representing the input image. Unlike other state-of-the-art methods, ours takes advantage

of the useful distortions of 360º panorama images and uses deep learning in two of its tasks. Besides, it does not require any pre-processing step, neither geometric nor contextual annotations in the input image, yet the results were exceptional, outperforming previous state-of-the-art methods.

Hopefully, this work will be continued, improved, and expanded to several other environments and application areas. For example, it can be adjusted to be used in Industry 4.0 to allow visualization of the shop floor and thus an easy way to control all the process, or be applied in virtual reality applications to permit a scene's automatic mapping. Another great improvement to implement in the future is the conversion of our pipeline into an end-to-end application.

In sum, this dissertation contributes to the 3D scene understanding area by systematically analyzing its state-of-the-art methods and creating a novel pipeline for 3D scene reconstruction that is overall efficient and accurate and a 3D scene visualization/adjustment application.

## 7.2 Contributions

Throughout this dissertation, several contributions were made. The most important ones are:

- **Literature Review of Scene Understanding's state-of-the-art:** We made a complete analysis of the object detection and the layout estimation tasks, comparing all the state-of-the-art methods of both tasks critically. Besides, we made a recap and a deep comparison of all the published 3D Scene Reconstruction methods, focusing on 360º panorama images. Lastly, we summarized all the available datasets of 360 panorama images.
- **A novel approach:** We proposed a complete approach for indoor 3D scene reconstruction based on 360º panorama images. Our approach combines deep learning in two of its phases and achieves outstanding results.
- **An application:** We developed a web/desktop application that allows the visualization and adjustment of the 3D scene automatically created.
- **Integration in the Critical Manufacturing MES solution:** Our solution was integrated into the FabLive component of the Critical Manufacturing MES, allowing the generation of 3D scenes and posterior visualization in its solution.

## 7.3 Difficulties

During the solution's development, there were several challenges to overcome in order to create a solution that better fitted the proposed constraints. Some of these difficulties were mentioned in the previous sections, some of them were solved during the solution's implementation, and others will be mentioned in section 7.4 (p. 81) as future work.

In the context of Industry 4.0, it was desirable to evaluate our solution in a dataset of panorama images that represented the shop floor context. However, due to the current pandemic situation, it was impossible to collect 360º panorama images from factories and perspective images of its

machines. Consequently, it was not possible to test our solution with this type of images, which could require some adjustments in the pipeline in order to obtain more accurate results.

In the 3D position reasoning and pose estimation phases, it would be advantageous for our pipeline to use a deep neural network to perform depth estimation since this would allow the resolution of two steps in just one. However, the use of this type of models requires complex labels for the images in the panorama dataset. These labels need, for instance, precise information about the object's measures and rotation angles, information that we did not have for that dataset. Despite this, a novel method was used to answer the 3D position reasoning and pose estimation phases' requirements, and as we can see in sections 5.3 (p. 65) and 5.4 (p. 66), the results were very accurate and satisfactory.

Another less critical difficulty that we face during the solution's implementation was the lack of computing resources. For the layout estimation and object detection phases, we needed to use the Google Colab service [8] to train and test our networks in order to access a GPU. Despite both the training and testing of the models have been successful, the time needed to perform was extremely long due to the Google Colab instability and because of the usage's limitations.

## 7.4 Future Work

The solution developed during this dissertation solved the most important issues identified in section 4.1 (p. 42) and ensures the main objective of this thesis. However, our implementation contains some limitations and introduces new challenges, which can be addressed and explored as future work.

The problem, previously discussed in section 7.3 (p. 80), concerning the lack of panorama images of the shop floor, can be addressed as future work. In principle, as soon as the restrictions cause by the global pandemic end, we can visit the installations of some factories and thus capture several panorama images of their shop floors. These images would allow the evaluation of our pipeline in the industry context, and consequently, if necessary, the proposed pipeline's change or adjustment to obtain better results.

Regarding the phases of 3D position reasoning and pose estimation, it would be very interesting both at a scientific/research level and for improving our application, experiment using depth estimation models based on deep learning. For that to be possible to carry out, as explained in section 7.3 (p. 80), we will need to create at least a small indoor panorama dataset with the required labels to train, test, and evaluate the depth estimation models.

Throughout the development of our solution, we noticed that to obtain a realistic 3D scene it is necessary to have realistic 3D models to represent each object. When the number of different objects in the scenes becomes very big, the work and time associated with creating their respective 3D models become impracticable. So, it would be very advantageous to have the possibility to generate a 3D model from several photos of the object in question. Currently, there are several photogrammetry software tools, many of which should have an API that enables the integration in our pipeline.

After testing our pipeline in large rooms/warehouses, we found that the object detection phase is quite compromised by not detecting the farthest away objects. To solve this problem, It would be useful to combine two panoramic images taken in opposite places in the room, to improve the object detection phase and consequently improve the final results.

In the future, we also aim to convert our pipeline into an end-to-end solution, to enhance usability and reduce computation time.

In sum, we can conclude that the developed tool has room for improvement, not only in the expansion to new environments but also in its optimization and enrichment. Furthermore, further experiments could be carried out to improve our pipeline and increase the scientific knowledge in the 3D scene understanding area.

# References


[1] 3D Warehouse. Available at https://3dwarehouse.sketchup.com/, accessed 2020-12-30.

[2] An example of a ROC curve. Available at https://towardsdatascience.com/a-simple-explanation-of-the-roc-curve-and-auc-64db32d75541, accessed 2020-12-15.

[3] COCO challenge. Available at https://cocodataset.org/{#}detection-eval, accessed 2020-12-16.

[4] Confusion Matrix. Available at https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62, accessed 2020-12-15.

[5] Convolutional Neural Networks architectures. Available at https://paperswithcode.com/method/resnet, accessed 2020-12-15.

[6] Example of IoU values for different bounding boxes. Available at https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, accessed 2020-12-15.

[7] Geometric projections in panoramic photography. Available at https://www.panoramic-photo-guide.com/geometric-projections.html, accessed 2021-06-01.

[8] Google Colab. Available at https://colab.research.google.com/, accessed 2021-03-08.

[9] "Little Planet" Photographs. Available at http://paulbourke.net/panorama/littleplanet/, accessed 2021-06-01.

[10] Multilayer perceptron model - example. Available at = https://github.com/rcassani/mlp-example, accessed 2020-12-15.

[11] Panorama formats. Available at https://wiki.panotools.org/Panorama_formats, accessed 2021-06-01.

[12] PANORAMIC IMAGE PROJECTIONS. Available at https://www.cambridgeincolour.com/tutorials/image-projections.htm#:~:text=An%20image%20projection%20occurs%20whenever,piece%20of%20paper%2C%20for%20example.l, accessed 2021-06-01.

[13] Precision-Recall curve - example. Available at https://www.researchgate.net/figure/Example-for-a-a-ROC-plot-and-b-a-precision-recall-curve-consensus-score-, accessed 2020-12-16.

[14] RNN architectures. Available at https://deepai.org/machine-learning-glossary-and-terms/recurrent-neural-network, accessed 2020-12-15.

[15] The Industrial Revolution from Industry 1.0 to 5.0. Available at https://supplychaingamechanger.com/the-industrial-revolution-from-industry-1-0-to-industry-5-0/, accessed 2020-12-14.

[16] The PASCAL Visual Object Classes Challenge 2007. Available at http://host.robots.ox.ac.uk/pascal/VOC/voc2007/index.html, accessed 2020-12-16.

[17] Understanding of Convolutional Neural Network (CNN) — Deep Learning. Available at https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148, accessed 2020-12-14.

[18] Iro Armeni, Alexander Sax, Amir R. Zamir, and Silvio Savarese. Joint 2D-3D-semantic data for indoor scene understanding. *arXiv*, 2017.

[19] Alexey Bochkovskiy, Chien Yao Wang, and Hong Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv*, 2020.

[20] Federico Boniardi, Abhinav Valada, Rohit Mohan, Tim Caselitz, and Wolfram Burgard. Robot Localization in Floor Plans Using a Room Layout Edge Extraction Network. In *IEEE International Conference on Intelligent Robots and Systems*, 2019.

[21] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[22] Ricardo Cabral and Yasutaka Furukawa. Piecewise planar and compact floorplan reconstruction from images. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.

[23] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niebner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. In *Proceedings - 2017 International Conference on 3D Vision, 3DV 2017*, 2018.

[24] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. *arXiv preprint arXiv:1512.03012*, 2015.

[25] Jiacheng Chen, Chen Liu, Jiaye Wu, and Yasutaka Furukawa. Floor-sp: Inverse cad for floorplans by sequential room-wise shortest path. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[26] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3), 1995.

[27] James M. Coughlan and A. L. Yuille. Manhattan World: Compass direction from a single image by Bayesian inference. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, 1999.

[28] Critical Manufacturing. MES FOR INDUSTRY 4.0. Available at https://www.criticalmanufacturing.com/en/critical-manufacturing-mes/complete-modular-solution, accessed 2020-12-14.

[29] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, volume I, 2005.

[30] Enes Dayangac and Gangolf Hirtz. Object recognition for human behavior analysis. In *IEEE International Conference on Consumer Electronics - Berlin, ICCE-Berlin*, volume 2015-February, 2015.

[31] Luca Del Pero, Joshua Bowdish, Bonnie Kermgard, Emily Hartley, and Kobus Barnard. Understanding bayesian rooms using composite 3D object models. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2013.

[32] Erick Delage, Honglak Lee, and Andrew Y. Ng. A dynamic Bayesian network model for autonomous 3d reconstruction from a single indoor image. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, 2006.

[33] Fucheng Deng, Xiaorui Zhu, and Jiamin Ren. Object detection on panoramic images based on deep learning. In *2017 3rd International Conference on Control, Automation and Robotics, ICCAR 2017*, 2017.

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[35] Nikita Dvornik, Konstantin Shmelkov, Julien Mairal, and Cordelia Schmid. BlitzNet: A Real-Time Deep Network for Scene Understanding. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-Octob, 2017.

[36] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008.

[37] Clara Fernandez-Labrador, Jose M. Facil, Alejandro Perez-Yus, Cedric Demonceaux, Javier Civera, and Jose J. Guerrero. Corners for Layout: End-to-End Layout Recovery from 360 Images. *IEEE Robotics and Automation Letters*, 5(2), 2020.

[38] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting BT - Computational learning theory. *Computational learning theory*, 904(Chapter 2), 2005.

[39] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[40] Germany Trade & Invest. Smart manufacturing for the future. Available at https://www.academia.edu/21125581/SMART_MANUFACTURING_FOR_THE_FUTURE_INDUSTRIE_4_0_Future_Markets, accessed 2020-12-14.

[41] Veta Ghenescu, Roxana Elena Mihaescu, Serban Vasile Carata, Marian Traian Ghenescu, Eduard Barnoviciu, and Mihai Chindea. Face Detection and Recognition Based on General Purpose DNN Object Detector. In *2018 13th International Symposium on Electronics and Telecommunications, ISETC 2018 - Conference Proceedings*, 2018.

[42] Ross Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:1440–1448, 2015.

[43] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.

[44] Clement Godard, Oisin Mac Aodha, Michael Firman, and Gabriel Brostow. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-October, 2019.

[45] Julia Guerrero-Viu, Clara Fernandez-Labrador, Cedric Demonceaux, and Jose J. Guerrero. What's in my Room? Object Recognition on Indoor Panoramic Images. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2020.

[46] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *2017 IEEE International Conference on Computer Vision*, 42(2), 2017.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9), 2015.

[49] Varsha Hedau, Derek Hoiem, and David Forsyth. Recovering the spatial layout of cluttered rooms. In *Proceedings of the IEEE International Conference on Computer Vision*, 2009.

[50] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[51] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8), 1982.

[52] Satoshi Ikehata, Hang Yang, and Yasutaka Furukawa. Structured indoor modeling. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[53] Hamid Izadinia, Qi Shan, and Steven M. Seitz. IM2CAD. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-January, 2017.

[54] Henning Kagermann, Reiner Anderl, Jurgen Gausemeier, and Gunther Wahlster Schuh. Industrie 4.0 in a Global Context: Strategies for Cooperating with International Partners (acatech STUDY. 2016.

[55] Hansung Kim, Luca Hernaggi, Philip J.B. Jackson, and Adrian Hilton. Immersive spatial audio reproduction for VR/AR using room acoustic modelling from 360° images. In *26th IEEE Conference on Virtual Reality and 3D User Interfaces, VR 2019 - Proceedings*, 2019.

[56] Pileun Kim, Jingdao Chen, and Yong K. Cho. SLAM-driven robotic mapping and registration of 3D point clouds. *Automation in Construction*, 89, 2018.

[57] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[58] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10), 2006.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.

[60] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *IJCV*, 2020.

[61] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

[62] Chen Yu Lee, Vijay Badrinarayanan, Tomasz Malisiewicz, and Andrew Rabinovich. RoomNet: End-to-End Room Layout Estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-Octob, Salt Lake City, UT, USA, 2017. IEEE.

[63] David C. Lee, Abhinav Gupta, Martial Hebert, and Takeo Kanade. Estimating spatial layout of rooms using volumetric reasoning about objects and surfaces. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010, NIPS 2010*, 2010.

[64] David C. Lee, Martial Hebert, and Takeo Kanade. Geometric reasoning for single image structure recovery. *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*, 2009 IEEE:2136–2143, 2009.

[65] Buyu Li, Wanli Ouyang, Lu Sheng, Xingyu Zeng, and Xiaogang Wang. GS3D: An efficient 3D object detection framework for autonomous driving. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2019-June, 2019.

[66] Mingyang Li, Yi Zhou, Ming Meng, Yuehua Wang, and Zhong Zhou. 3D Room Reconstruction from A Single Fisheye Image. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2019-July, 2019.

[67] Rainer Lienhart and Jochen Maydt. An extended set of Haar-like features for rapid object detection. In *IEEE International Conference on Image Processing*, volume 1, 2002.

[68] Tsung Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-January, 2017.

[69] Tsung Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal Loss for Dense Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2), 2020.

[70] Chenxi Liu, Alexander G. Schwing, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Rent3D: Floor-plan priors for monocular layout estimation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June-2015, 2015.

[71] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation, 2018.

[72] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9905 LNCS, 2016.

[73] Zicheng Liu, Yan Zhang, Wentao Wu, Kai Liu, and Zhengxing Sun. Model-driven indoor scenes modeling from a single image. In *Proceedings - Graphics Interface*, volume 2015-June, 2015.

[74] Stuart P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[75] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2), 2004.

[76] Xiangsong Meng, Xiangli Zhang, Kun Yan, and Hongmei Zhang. Real-Time Detection and Recognition of Live Panoramic Traffic Signs Based on Deep Learning. In *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, volume 2018-November, 2019.

[77] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. volume 1, 2009.

[78] Giovanni Pintore, Valeria Garro, Fabio Ganovelli, Enrico Gobbetti, and Marco Agus. Omnidirectional image capture on mobile devices for fast automatic generation of 2.5D indoor maps. In *2016 IEEE Winter Conference on Applications of Computer Vision, WACV 2016*, 2016.

[79] Lemonia Ragia, Froso Sarri, and Katerina Mania. 3D reconstruction and visualization of alternatives for restoration of historic buildings a new approach. In *GISTAM 2015 - 1st International Conference on Geographical Information Systems Theory, Applications and Management, Proceedings*, 2015.

[80] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-December, 2016.

[81] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, 2017.

[82] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv*, 2018.

[83] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, volume 2015-Janua, 2015.

[84] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 1958.

[85] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088), 1986.

[86] Berkman Sahiner, Aria Pezeshk, Lubomir M. Hadjiiski, Xiaosong Wang, Karen Drukker, Kenny H. Cha, Ronald M. Summers, and Maryellen L. Giger. Deep learning in medical imaging and radiation therapy, 2019.

[87] Alexander G. Schwing, Tamir Hazan, Marc Pollefeys, and Raquel Urtasun. Efficient structured prediction for 3D indoor scene understanding. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2012.

[88] Alexander G. Schwing and Raquel Urtasun. Efficient exact inference for 3D indoor scene understanding. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7577 LNCS, 2012.

[89] Tianjia Shao, Weiwei Xu, Kun Zhou, Jingdong Wang, Dongping Li, and Baining Guo. An interactive approach to semantic modeling of indoor scenes with an RGBD Camera. In *ACM Transactions on Graphics*, volume 31, 2012.

[90] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–14, 2015.

[91] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding LSTM – A tutorial into Long Short-Term Memory Recurrent Neural Networks. *arXiv*, pages 1–42, 2019.

[92] Cheng Sun, Chi Wei Hsiao, Min Sun, and Hwann Tzong Chen. Horizonnet: Learning room layout with 1d representation and pano stretch data augmentation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2019-June, Long Beach, CA, USA, USA, 2019. IEEE.

[93] Koen E.A. Van De Sande, Jasper R.R. Uijlings, Theo Gevers, and Arnold W.M. Smeulders. Segmentation as selective search for object recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, 2011.

[94] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, 2001.

[95] G. Vosselman. 3D Reconstruction of Roads and Trees for City Modelling. *ISPRS Workshop*, 34, 2003.

[96] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn, 2019.

[97] Jianxiong Xiao, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Recognizing scene viewpoint using panoramic place representation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2012.

[98] Jiu Xu, Bjorn Stenger, Tommi Kerola, and Tony Tung. Pano2CAD: Room layout from a single panorama image. In *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, Santa Rosa, CA, USA, 2017. IEEE.

[99] Hao Yang and Hui Zhang. Efficient 3D room shape recovery from a single panorama. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-December, 2016.

[100] Shang Ta Yang, Fu En Wang, Chi Han Peng, Peter Wonka, Min Sun, and Hung Kuo Chu. DuLa-Net: A dual-projection network for estimating room layouts from a single RGB panorama. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2019-June, 2019.

[101] Wenyan Yang, Yanlin Qian, Joni Kristian Kamarainen, Francesco Cricri, and Lixin Fan. Object Detection in Equirectangular Panorama. In *Proceedings - International Conference on Pattern Recognition*, volume 2018-August, 2018.

[102] Yang Yang, Shi Jin, Ruiyang Liu, Sing Bing Kang, and Jingyi Yu. Automatic 3D Indoor Scene Modeling from Single Panorama. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018.

[103] Wei Zeng, Sezer Karaoglu, and Theo Gevers. Pano2Scene : 3D Indoor Semantic Scene Reconstruction from a Single Panorama Image. 2020.

[104] Jian Zhang, Chen Kan, Alexander G. Schwing, and Raquel Urtasun. Estimating the 3D layout of indoor scenes and its clutter from depth sensors. In *Proceedings of the IEEE International Conference on Computer Vision*, 2013.

[105] Yinda Zhang, Shuran Song, Ping Tan, and Jianxiong Xiao. PanoContext: A whole-room 3D context model for panoramic scene understanding. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8694 LNCS, 2014.

[106] Yibiao Zhao and Song Chun Zhu. Scene parsing by integrating function, geometry and appearance models. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2013.

[107] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34, 2020.

[108] Keliang Zhou, Taigang Liu, and Ling Liang. From cyber-physical systems to Industry 4.0: Make future manufacturing become possible. *International Journal of Manufacturing Research*, 11(2), 2016.

[109] Chuhang Zou, Alex Colburn, Qi Shan, and Derek Hoiem. LayoutNet: Reconstructing the 3D Room Layout from a Single RGB Image. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2051–2059, Salt Lake City, UT, USA, 2018. IEEE.

# Appendix A

# JSON file generated by our pipeline

This appendix presents an example of a JSON file generated by our pipeline after all the phases described in chapter 4 and 5. This file will also be the input to our web/desktop application that will create the 3D scene automatically.

```json
{
    "img_path": "demo_aligned_rgb.png",
    "dimensions": {
        "x": "2.9097135",
        "y": "1.6",
        "z": "4.0935926"
    },
    "floor_color": "6f321a",
    "walls_color": "c9ac6d",
    "walls": [
        {
            "from": {
                "x": "-0.1959936",
                "z": "2.0545576"
            },
            "to": {
                "x": "-0.19599354",
                "z": "3.9839153"
            },
            "height": "1.6"
        },
        {
            "from": {
                "x": "-0.19599354",
                "z": "3.9839153"
            },
            "to": {
                "x": "-1.3569014",
                "z": "3.9839144"
            },
            "height": "1.6"
        },
        {
            "from": {
                "x": "-1.3569014",
                "z": "3.9839144"
            },
            "to": {
                "x": "-1.3569022",
                "z": "-4.093592"
            },
            "height": "1.6"
        },
        {
            "from": {
                "x": "-1.3569022",
                "z": "-4.093592"
            },
            "to": {
                "x": "2.9097128",
                "z": "-4.0935926"
            },
            "height": "1.6"
        },
        {
            "from": {
                "x": "2.9097128",
                "z": "-4.0935926"
            },
```

```
            "to": {
                "x": "2.9097135",
                "z": "1.1095303"
            },
            "height": "1.6"
        },
        {
            "from": {
                "x": "2.9097135",
                "z": "1.1095303"
            },
            "to": {
                "x": "1.0311482",
                "z": "1.1095303"
            },
            "height": "1.6"
        },
        {
            "from": {
                "x": "1.0311482",
                "z": "1.1095303"
            },
            "to": {
                "x": "1.0311477",
                "z": "2.0545578"
            },
            "height": "1.6"
        },
        {
            "from": {
                "x": "1.0311477",
                "z": "2.0545578"
            },
            "to": {
                "x": "-0.1959936",
                "z": "2.0545576"
            },
            "height": "1.6"
        }
    ],
    "floor": [
        [
            "-0.1959936",
            "2.0545576"
        ],
        [
            "-0.19599354",
            "3.9839153"
        ],
        [
            "-1.3569014",
            "3.9839144"
        ],
        [
            "-1.3569022",
            "-4.093592"
        ],
        [
            "2.9097128",
            "-4.0935926"
        ],
```

```
            [
                "2.9097135",
                "1.1095303"
            ],
            [
                "1.0311482",
                "1.1095303"
            ],
            [
                "1.0311477",
                "2.0545578"
            ]
        ],
        "wallsTextureUV": [
            [
                "0.0146484375",
                "0.3008330762386322"
            ],
            [
                "0.0146484375",
                "0.7089354991912842"
            ],
            [
                "0.007335239555686712",
                "0.38581281900405884"
            ],
            [
                "0.007335239555686712",
                "0.6204522848129272"
            ],
            [
                "0.0517578125",
                "0.3912762403488159"
            ],
            [
                "0.0517578125",
                "0.6146637797355652"
            ],
            [
                "0.4485706090927124",
                "0.3936861753463745"
            ],
            [
                "0.4485706090927124",
                "0.6121071577072144"
            ],
            [
                "0.5978592038154602",
                "0.4077087640762329"
            ],
            [
                "0.5978592038154602",
                "0.597193717956543"
            ],
            [
                "0.8074917793273926",
                "0.35766440629959106"
            ],
```

```
        [
            "0.8074917793273926",
            "0.6501006484031677"
        ],
        [
            "0.8803366422653198",
            "0.2525349259376526"
        ],
        [
            "0.8803366422653198",
            "0.7577382922172546"
        ],
        [
            "0.925480306148529",
            "0.3167843818664551"
        ],
        [
            "0.925480306148529",
            "0.6925708055496216"
        ]
    ],
    "objects": [
        {
            "id": "Bedroom-chair",
            "color": "rgb(28, 199, 73)",
            "position": {
                "x": -1.1073566371420098,
                "y": 0,
                "z": 1.9642648283699249
            },
            "size": {
                "x": 0.34871378317307694,
                "y": 0.6,
                "z": 0.34871378317307694
            },
            "rotation": "-90"
        },
        {
            "id": "Bedroom-chair",
            "color": "rgb(28, 199, 73)",
            "position": {
                "x": -0.9588865176490072,
                "y": 0,
                "z": -1.0669579488217364
            },
            "size": {
                "x": 0.34871378317307694,
                "y": 0.6,
                "z": 0.34871378317307694
            },
            "rotation": "-90"
        },
        {
            "id": "Table",
            "color": "rgb(255, 145, 0)",
            "position": {
                "x": -0.8301274255015487,
                "y": 0,
                "z": -2.1113910113619445
            },
```

```
            "size": {
                "x": 1.2204982411057692,
                "y": 0.4,
                "z": 0.7179401418269231
            },
            "rotation": "90"
        },
        {
            "id": "Bed",
            "color": "rgb(227, 23, 193)",
            "position": {
                "x": 1.674652664663462,
                "y": 0,
                "z": -2.463108965564904
            },
            "size": {
                "x": 1.9486946706730768,
                "y": 0.35,
                "z": 1.0717820688701922
            },
            "rotation": "90"
        }
    ]
}
```