UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO

Deep Learning based self-localization in virtual environments

Tiago João Barbosa Pereira

Mestrado em Engenharia Informática

Trabalho de Projeto orientado por:
Professor Doutor Thibault Nicolas Langlois

2025

Aos meus pais, por tudo o que eu sou e serei.

Agradecimentos

Se há alguma coisa que aprendi durante a minha vida universitária é de como nunca se faz as coisas completamente sozinho, e que nada pode substituir o apoio que os que te rodeiam podem oferecer.

Agradeço à minha família por terem criado a pessoa que sou hoje, me terem dado a possibilidade de estudar o que eu queria e que sempre apoiaram financeiramente e moralmente as minhas escolhas académicas e pessoais.

Agradeço aos meus amigos, por terem me terem acompanhado desde o início, ajudado no que eu precisava, e me fazerem lembrar que existe vida para ser experienciada fora da tese e para não me limitar por ela.

Agradeço à minha namorada pelo apoio emocional e imensa paciência, que me sempre incentivou a ser melhor e a nunca desistir nos momentos mais difíceis.

Agradeço ao meu orientador, professor Thibault Langlois pelo apoio imprescindível, a disponibilidade, o acompanhamento e a preocupação que ajudou muito durante todo este percurso.

Um profundo obrigado pelo que estas pessoas fizeram por mim, levarei comigo para sempre as experiências e ensinamentos que me proporcionaram.

Obrigado por tudo.

Tiago Pereira

Resumo

Aprendizagem automática é um meio crucial para alcançar soluções rápidas e eficientes para problemas difícilmente resolvidos por humanos, como por exemplo reconhecimento de imagens, análise preditiva, diagnósticos médicos automáticos e auto-localização.

Esta última área de investigação mencionada é o foco deste projeto, onde foram exploradas técnicas de aprendizagem automática para o problema de auto-localização em ambientes artificiais 3D, recorrendo a arquiteturas supervisionadas de aprendizagem automática baseadas em imagens, nomeadamente VGG (*Visual Geometry Group*), que é uma arquitetura de rede neuronal convolucional (CNN).

A motivação para a utilização de ambientes virtuais neste projeto é explorar o problema de dependência de dados presente no treino de qualquer modelo de aprendizagem automática, onde a qualidade deste conjunto de dados afeta imensamente o desempenho do modelo. Uma dos principais cuidados deste projeto foi manter um nível alto de qualidade e garantir devida etiquetação dos dados. Para este fim, é usado o programa de modelação 3D *Blender* para a criação de ambientes virtuais onde vão ser geradas imagens que constituem os datasets destinados especificamente ao treino de redes convolucionais. Recorrendo à API de *Python* embutida no *Blender*, os ambientes são construídos através de programação orientada a objetos, que permite criar e modificar objetos tridimensionais no ambiente virtual, e obter a informação necessária para etiquetar corretamente os dados gerados.

As primeiras cenas construídas representam salas cúbicas com objetos, espalhados pelo chão ou distribuídos por estantes, e paredes cinzentas sem qualquer fator distintivos entre elas. O modelo tem como objetivo prever a sua localização baseada apenas em imagens geradas nestas cenas.

Os datasets são criados com a preocupação de manter qualidade, uma vez que esta qualidade se traduz num melhor desempenho do modelo. Alguns aspetos que têm de ser respeitados para obter um nível alto de qualidade são os seguintes: acessibilidade, credibilidade, consistência e usabilidade. A utilização de ambientes virtuais contruídos no Blender, criados especialmente para a tarefa de localização permite alcançar estes aspetos de forma simplificada e eficiente.

As técnicas usadas para gerar as imagens de treino e teste do modelo têm de principalmente capturar o ambiente de forma clara e completa. A primeira versão do processo de captura de imagens consiste em mover a câmera para pontos equidistantes, com uma distribuição por grelha. Em cada uma destas localizações, a câmera gera imagens em torno dela, criando uma imagem a cada determinado ângulo de rotação. A distribuição por grelha dos pontos de captura provou-se

limitante, sendo concebida uma distribuição baseada na sequencia de Sobol que espalha de forma mais completa estes pontos por toda a área de captura. Por cada imagem é gerado também um ficheiro JSON que guarda informação sobre a imagem, como a sua localização, ângulo de rotação e objetos visíveis. Estas informações são usadas para etiquetar as imagens com o intuito de serem aplicadas no treino supervisionado do modelo.

Um pequeno teste foi realizado para determinar as dimensões da resolução e ângulo campo de visão (FOV) que iriam descrever as imagens, de maneira a capturarem claramente o ambiente e os seus detalhes. Uma resolução baixa de 128x128 foi escolhida, uma vez que os ambientes usados não possuem grandes detalhes que podessem ser omitidos pela resolução, e um FOV de 36°, valor padrão do Blender que permite capturar a cena e os seus objetos sem grande distorção da imagem.

A rede convolucional usada é baseada na arquitetura VGG, que consiste em blocos de convolução formados por: duas camadas de convolução seguidos da função de ativação ReLU, e termina com uma camada de maxPooling. Estes blocos de camadas realizam operações de convolução sobre os dados de entrada, envolvendo multiplicação elemento a elemento entre o filtro e uma dada região da imagem de entrada, resultando em *feature maps* que destacam características desta, como cores, formas e partes de objetos. As camadas de *MaxPooling* diminuem as dimensões destes mapas de forma a reduzir a quantidade de parâmetros a serem computados, enquanto mantem as informações mais relevantes. A arquitetura do modelo é constituída por quatro blocos de convolução e termina com três camadas *fully-connected*. Devido ao número de camadas/profundidade que possui, a arquitetura desenvolvida pode se considerar uma VGG11.

Foram concebidas duas formas diferentes de localização, uma que aplica classificação e outra regressão. A classificação serve como uma prova de conceito, que identifica zonas de localização quadradas que formam uma grelha de zonas, e que as trata como classes a serem previstas, e é por isso uma localização simples e pouco precisa. A regressão procura localizar exatamente as coordenadas e ângulo de rotação em que foram geradas as imagens, e é esta a versão definitiva dos modelos estudados.

O método de treino usado inicialmente consiste em usar folds, em que a determinado número de épocas o dataset de treino e validação altera. Este método foi concebido para conseguir evitar overfitting e permitir que a geração de dados e treino do modelo possa acontecer simultaneamente.

Nas experiências da classificação, com uma grelha que cobre todo o ambiente de dimensões 3x3, ou seja, 3 classes para a coordenada X e 3 para a Y, foram obtidos resultados desapontantes, com uma precisão de 56.73% num problema de baixa exatidão devido ao tamanho de cada zona. Um fator limitante foram imagens que não capturam elementos distintivos, causando demasiado ruído durante o treino do modelo. Para verificar e solucionar este problema, foi aplicada uma filtragem no processo de renderizar imagens, de maneira a criar datasets com apenas imagens em que se deteta objetos relevantes. Os resultados melhoraram para os 94.18%, quando se manteve as mesmas condições, mas quando se aumentou a dimensão da grelha para 32x32, a precisão obtida foi de 50.69%, mesmo quando aplicando o filtro de objetos.

As seguintes experiências aplicam regressão, em que não só foram realizadas mudanças ao

modelo, mas também ao ambiente. Nesta fase são previstas a posição absoluta e ângulo das imagens, por isso espera-se obter uma precisão melhor em relação à classificação. Foram feitos experiências que estudam o benifcio do filtro de objetos, onde ainda se comprova a vantagem deste fator, sendo que nas mesmas condições, a distância média entre os valores reais e previstos melhorou de 50.41 para 7.90 unidades métricas do Blender(as unidades dos resultados expostos são omitidas).

Os datasets gerados deixaram de refletir o ambiente na sua totalidade por causa da utilização do filtro de objetos, sendo por isso concebido outro método, em que um par de imagens (stereo) é usado como input do modelo, cada com uma vista de 180°oposta, de maneira a que o modelo receba mais contexto espacial sobre as imagens, podendo ter sido capturado em pelo menos uma delas pontos referenciais suficientes para localização. Os resultados obtidos não foram ao encontro do esperado, havendo grandes picos de perda ao longo do treino devido à troca de *folds*, em que cada *fold* corresponde a um dataset em que o modelo treina durante um dado número de épocas.

Foi aplicado outro método de treino, em que foram gerados previamente os datasets necessários, e as imagens são geradas em batches que podem ser adicionadas ao longo do treino. Este método provou superar o anterior, alcançando distâncias de 34.35 para input singular e 39.94 para stereo.

Mais estudos sobre a influência das mudanças no ambiente foram exploradas, como adicionar elementos que servem de marcadores, e pintar as paredes do ambiente com cores distintas para serem distinguíveis. Este último provou ser especialmente benéfico, mesmo quando alterando ligeiramente a tonalidade das paredes, alcançando a melhor distância média de erro de 2.83. Contudo, em todas as experiências realiadas, a versão stereo nunca superou o desempenho da versão de *input* singular.

Concluindo, a versão do modelo que recebe como input o par de imagens, apesar de apresentar resultados satisfatórios em alguns cenários, apresenta um desempenho mais fraco comparativamente com a versão que recebe apenas uma imagem. Apesar disto, este trabalho apresentou várias experiências que exploram a influência de elementos adicionais no ambiente, dos quais forma obtidas conclusões interessantes, como o facto de alterar ligeiramente a cor das paredes melhorou muito o desempenho do modelo. Direções futuras para este trabalho foram também apresentadas, demonstrando que os tópicos discutidos e trabalho desenvolvido requerem continuar a ser explorados para alcançar localização em ambientes artificiais, para possível aplicação no mundo real.

Palavras-chave: Deep Learning, Localização, Blender, Rede Neuronal Convolucional, VGG

Abstract

Machine learning systems can greatly improve tasks that are otherwise difficult, one of which is self-localization. This thesis explores the use of *Convolutional Neural Networks* to achieve self-localization in 3D virtual environments by developing a supervised learning model based on the VGG CNN architecture. The virtual 3D environments are created using the 3D modeling program *Blender*, which includes a Python API that enables the scene to be constructed entirely in code. Resorting to this process addresses one of the primary issues common to every machine learning algorithm: the acquisition of reliable datasets that accurately represent the problem's dimensions, along with accurately labeled items. The easily modifiable environment helps in the study of the impact of the scene on the performance of the localization model. The experiments conducted explored a VGG model that receives as input a pair of images (stereo) that capture opposite views of the scene, as an attempt to illustrate more environmental context. These experiments were also conducted with a single image as input version of the model, to compare the performances. The stereo iteration, while providing acceptable results in some environments, demonstrated to be under-performing compared to the single version in every scene that was tested. The experiments also demonstrated the influence that the datasets have over the model's performance, where changes to the environments that build these datasets were needed to achieve decent results. Future work that further explores this discrepancy was also presented.

Keywords: Deep Learning, Localization, Blender, Convolutional Neural Network, VGG

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning presents itself as a crucial means to achieve more efficient and faster solutions to problems that are difficult for humans to resolve. These algorithms are designed to enable computers to learn from the data gathered and make decisions based on the results obtained, allowing tasks such as image recognition, predictive analysis, automated medical diagnosis, natural language processing, and self-localization.

This last branch of research is the focus of this project, where algorithms capable of self-localization in 3D artificial environments are developed and studied, resorting to supervised image-based machine learning architectures, namely VGG.

1.1 Motivation

In this project, one of the problems that is being explored is the dataset and how difficult it is to obtain and/generate data with the correct labels, which is an issue present in every sector that employs ML.

The results are obtained through patterns found in the data used for training, so presenting a way to simplify data gathering is beneficial and relevant to this field, meaning it is one of the main difficulties found when employing this technology. The quality and quantity of data utilized directly influence how the model learns and how accurate its predictions are. Datasets need to be formed of diverse and representative data that illustrate the problem at hand to better enhance the recognition of patterns, reduce biases, and improve performance when applied to real-world problems [6]. To that effect, different methods of image capturing, input representation, and environmental changes will be employed to further explore the dataset's influence over the model and how to improve them to obtain a better model's performance for the task at hand.

The program *Blender* will be used to reliably obtain data to specifically construct image datasets. *Blender* is a 3D modeling program that provides tools for image and video rendering, animation, and compositing, being a complete set of tools for professionals in fields such as animation, video games, and visual effects. As such, the program is capable of generating 3D environments that can be used to render sets of images that can be used as datasets for machine learning. In section 3.1, the advantages of using this program are presented to illustrate why this

program was chosen to generate the learning datasets.

Using the program's built-in *Python* API, scenes will be built utilizing an Object-Oriented approach, where each element of the scene is associated with a *Python* object, including the scene itself. This approach was chosen to easily change environmental elements using code only. The scenes will depict a simple square room with objects scattered, in which the model will attempt to locate itself based solely on visual information gathered from the rendered images of the dataset.

The code developed in the project "An API For Building Artificial Worlds For Machine Learning Using Blender"[12] was used as a basis for this project. It is a thesis project that uses *Blender*'s *Python* API to build 3D environments. Several improvements to the environmental building and specifications for machine learning purposes were added to this project, which will be explained in later chapters.

1.2 Goals

The goal of this project is to successfully develop an image-based machine learning algorithm capable of self-localization in different 3D scenes constructed in *Blender*, that explores the addition of elements that may help identify location or add noise to the learning process. This artificial approach is flexible and easily modifiable, which is beneficial for analyzing the results in a possible real-world case, testing the model's weaknesses, and gaining a deeper understanding of how they arise and how to overcome them. If proven to be a viable method, then the problem related to data dependency is a less prominent issue for this type of exercise.

The work will focus on creating a neural network model, specifically a *Convolutional Neural Network* (CNN), that enables a robot to self-locate within a virtual 3D environment. This model will be trained exclusively on images generated within the simulated environment, with the primary goal of the robot being reliable and precise localization.

1.3 Structure of the document

The rest of the thesis follows the structure described in the following:

- Chapter 2 - Related Works

  Other works related to the topics being discussed are presented, given a brief description, and explained to their relevancy to the project at hand.

- Chapter 3 - Data Generation

  Describes in detail the part related to the generation of datasets, including the scripts that are used in *Blender* and the base scenes where the images are rendered.

- Chapter 4 - Machine Learning Model

The machine learning algorithm, a VGG-inspired *Convolutional Neural Network*, that will be used to learn from the datasets created in the previous chapter, is demonstrated and explained here.

- Chapter 5 - Training and Results

  The various training conditions and scenarios are established alongside the results obtained in the experiments. These results are explained as they are presented, since their quality influences the evolution of the training process and direction of the experiments.

- Chapter 6 - Future Work

  Possible directions that can be followed to continue the topics explored in this thesis.

- Chapter 7 - Conclusion

  Final thoughts and concluding comments about the work done, as well as a summary of the conclusions obtained.

Chapter 2

Related Work

This chapter presents state-of-the-art studies related to the themes explored in this thesis. The relevance of each work referenced is explained in terms of how it relates to the project being done. It is divided into three sections: 1) Machine learning architectures 2.1, 2) Localization techniques 2.2, 3) Dataset Quality 2.2

2.1 Machine Learning Architectures

In this section, works related to localization machine learning architecture are going to be presented and explained to illustrate the state-of-the-art, including VGG, which will be the inspiration for the model developed in this project.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a challenge that aims to evaluate image-based algorithms. This allows for researchers to contribute in this scientific field, further exploring their work to improve previously established state-of-the-art standards. [31].

This yearly challenge that started in 2010 consists of the classification of large-scale image datasets, with object recognition classified with 1000 labels. The achievements reached thanks to these projects turn into the benchmark of visual recognition machine learning.

One of these benchmark projects was conducted by Krizhevskii et al., who achieved the then-best error rate scores at the time, significantly improving on previous iterations of the challenge [19].

Another aspect that was tackled was the overfitting problem, which is very present in this type of study. Extensive training procedures can cause models to overfit, where in tests, the performance was less than desirable because the model would be adjusted to the learning dataset and would not be fit to predict the results of different data (test dataset). In 2022, [32] Claudio dos Santos et al. conducted a survey about regularization strategies for avoiding overfitting in Convolutional Neural Networks, where out of the selection of techniques analyzed, 27.6% of them are labeling modifications, 31.0% internal changes to the model, and the most common ones are input alterations, at 41.4%.

Manipulation of the dataset itself was the solution reached in [19], using a technique called data augmentation. Two dimensions were explored: the first was creating smaller patches that would translate inside the original images, creating a larger dataset, and horizontal rotations. The second improvement was altering the intensity of the RGB channels of the images, resulting in a characteristic that would translate to the model better identifying objects in natural images, since the identification of said objects shouldn't vary based on the light levels it is subjected to.

In 2014 , Karen Simonyan and Andrew Zisserman [33] explored further an important characteristic to Convolutional Neural Networks, it's depth, and how it influences the accuracy of models in a large-scale image recognition setting. By utilizing small convolutional filters at every layer, a further depth can be achieved. A 3x3 filter was the smallest possible size, able to capture spatial 3D notions (center, left/right, up/down). The basic structure of these experiments is a set of convolutional layers possessing ReLU non-linearity, followed by maxpooling. The number of layers at each set varies by experiments, which means different depths were explored to achieve the one that would obtain the better accuracy. At the end of the structure are three fully-connected layers, the first two have 4096 channels, and the last 1000, for that was the number of classes to be predicted in the experiments. While the other aspects would change between iterations of the model, these

ConvNet Configuration					
A	**A-LRN**	**B**	**C**	**D**	**E**
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 x 24 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					
Number of Parameters (in millions)					
133	133	133	134	138	144

Table 2.1: ConvNet(VGG) configurations and number of parameters adapted from tables of paper [33]: "The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity."

final layers would remain the same.

In 2.1 are the configurations that were explored, along with the number of weight layers, which is what is considered depth.

Utilizing a larger number of convolutional layers with smaller filters instead of a smaller quantity of layers with bigger filters, which translates to similar effective perception fields, results in applying a higher number of nonlinear rectifications overall. This also reduces the number of parameters, for example, comparing 3 3x3 convolutional layers with a certain C number of channels, that would correspond to $3 * (3^2 * C^2) = 27C^2$, while a comparatively same volume of perception fields 7x7 convolutional layer would correspond to $7^2 * C^2 = 49C^2$, a significant increase in weighs. The increase of depth also doesn't have a big impact on the number of parameters, as seen in table 2.1, where at the lowest depth, the number of parameters is 133 millions, and with the biggest depth only increases to 144 million.

VGG Version (ConvNet config)	Smallest image side		Top-1 val. error (%)	Top-5 val. error (%)
	Train (S)	Test(Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

Table 2.2: VGG results from paper [33]

The tests were conducted following previously established regulations in [19] in an attempt to obtain improved results. It consists on classification tasks of input images that try to predict the correct class from the 1000 possible classes. The evaluation measurements are the top-1 and top-5 errors, the former indicates the percentage of times the correct class was the one with the higher score, and the latter, which is the main evaluation method in the project, is if the correct class is present in the top 5 classes with the higher probability rates.

From the results obtained in the experiments and showcased in table 2.2, it was concluded that the classification error decreased as the depth increased, with model E being the one with the best score, with a depth of 19. In tests where the depth was the same but different architectures were explored (C and D), indicated that the extra non-linearity derived from the 3 1x1 convolutional layers wasn't as beneficial as a single 3x3 convolutional layer. This indicates that maintaining a certain level of spatial context from the convolutional layers' receptive fields is an important aspect for convolutional architectures.

Another aspect tested was applying scale jittering on the dataset images during the training project (the images would have values for width and height that would vary between 256 and 512). This variable scale proved to achieve a lower error percentage when evaluations were performed in a static dataset. Considering experiments ran with configuration E, the error percentages went from 9.0% and 8.7% when learning from static image datasets with 256x256 and 384x384 resolution, respectively, to 8.0% when learning from a variable-size image dataset. This part of the study contributes to the idea that dataset quality is vital for better performance from the machine learning model.

Overall, this experiment provided a method for improving the state-of-the-art convolutional

neural networks, expanding the depth of the model. The models explored in this project were named VGG, and it is annotated along with their depth, so the model with a depth of 19 learning layers is denominated VGG-19. The main characteristic of this architecture that distinguishes it from other CNNs is the consequential placement of convolutional layers [33] that allow an increased capacity to learn patterns from the data and allow the subsequent convolution layers to reuse and refine the feature maps produced.

VGG has been used in practical real-world cases, such as to categorize cases of dementia and to find early signs of the sickness, which is a difficult one to diagnose. The models developed achieved accuracies of up to 99%, proving to be very successful in this field [2].

2.2 Localization Methods

Image-based localization is a topic of interest for research communities and has generated a significant amount of research, focusing not only on various localization methods but also on different environments. This section demonstrates several works that use different architectures and inputs to achieve localization.

In 2019, Jiao et al. [17] conducted a survey on object detection, a dimension of machine learning localization, where several studies and state-of-the-art architectures are reviewed. The paper showcases trends in this area, indicating the direction and how this topic is evolving, such as the prominence of object detection through techniques that range from video object detection, multi-modal detection, salient object detection, etc., the usefulness in the medical field for diagnosis and monitoring biometrics, even the creation of artificial images that mix with real world image datasets to grow its size. This field of study can be approached from a lot of different angles with different objects that study its limits and how to surpass them.

A survey was conducted in 2020 [8] that analyzes the state-of-the-art deep learning localization and mapping. It divides the works into four categories: odometry estimation, mapping, global localization, and SLAM (Simultaneous Localization and Mapping). Odometry estimation calculates the change in pose, both translation and rotation, between frames of sensor data to track self-motion, which can be enhanced with deep learning by modeling dynamics and extracting features from the motion, improving accuracy. Mapping builds a representation of the environment, such as a visual 2D or 3D map, or even more abstract ones like a topological map that uses a graph-like representation of localization nodes connected by edges that represent transversable paths. Mapping aids in human and robot tasks and in correcting odometry errors. Deep learning benefits this area with techniques such as depth prediction, semantic labeling, and geometric reconstructions. Global localization discovers the position of a mobile agent in a scene by matching input data with a 2D or 3D map, reducing pose drift, and solving the kidnapped robot problem, in which a robot must navigate in an unknown environment without any information about the movement it must perform. Deep learning benefits this method by tackling complex data associated with changes in view and environmental differences, such as lighting and weather. SLAM incorporates the other three, boosting the performance of localization and mapping that could be obtained by using them separately.

The survey presents and discusses open questions, one being the comparison between end-to-end models and hybrid models. They explain that end-to-end models, which learn directly from raw data to perform their task without any intermediate steps, while they present a fast evolution that shows increasing performance in accuracy thanks to deep learning, are easily integrated with other learning tasks, forming hybrid models. Different from end-to-end models that only rely on deep learning frameworks, hybrid models integrate classical geometric models into the deep learning frameworks, like incorporating learned depth estimates in conventional visual odometry

algorithms to ”recover the absolute scale metric of poses”, which is a problem of scale ambiguity that surges from the image-based approach that can´t reconstruct the environment with a consistent global scale. Hybrid models already present great results in visual odometry and global localization. Another discussion topics is the multitude of possible sensors that can be used to capture the environment to create datasets, and how new types of sensors and configurations, and their combination into hybrid capture methods, are underexplored and may prove to be effective methods to achieve localization. Another point worth mentioning is the scalability of proposed models, where they may provide great accuracies in the environments where tests occurred, but may not be successful when tested in other scenarios. They provide the example of odometry estimations, which are normally tested in city areas and roads, but these results cannot be translated into good performance in rural or forest areas.

In 2023, another survey [9] about deep learning visual localization and mapping was conducted that analyzes the same type of works as the previous one, but with a focus on visual data. It analyses how deep learning can benefit visual localization in creating a general-purpose SLAM system, presenting three properties: 1) deep learning provides powerful perception tools that help extract features that are difficult to odometry estimations and global localization, and allow dense depth and semantic labeling for mapping; 2) deep learning offers scene understanding and interaction for robots, enabling high-level tasks such as retrieving objects from an environment; 3) deep learning permits SLAM systems and other localization algorithms to learn from past experience, self-learning from new information and adapting to achieve localization in new environments.

Another similar but important question is how deep learning can be applied to solve visual localization and mapping. Some of the more pertinent arguments are: 1) deep learning acts as a universal approximator to copy functions of localization and mapping algorithms; 2) deep learning can use its complex semantic labelling abilities to to solve association problems in SLAM, as in connecting images to positions in the environment; 3) deep learning can automatically detect task-relevant features; 4) deep learning can set up a self-learning system that automatically updates parameters based on the input images; 5) deep learning can solve more specific problems that are intrinsic with SLAM and localization/mapping algorithms, such as the monocular SLAM problem which can be solved with learned depth estimates with the absolute scale from deep learning.

A robot trolley called SmartTrolley was presented as a solution to localization in indoor environments, specifically a warehouse [4]. The robot is a mobile platform for moving heavy loads inside a real-world indoor warehouse that is equipped with two LiDARs (Light Detection And Ranging), which use lasers to measure distances, and wheel odometers that measure distances with ticks that occur at certain wheel rotation angles. For each LiDAR, they implemented an optimization-based scan-matching algorithm called NDT-PSO to enable SLAM to simultaneously localize itself and map the environment. The SLAM system uses an EKF fusion to fuse the poses obtained from the LiDAR sensors and the wheel encoders. The proposed solution was a success, resulting in Mean Absolute Error values in the centimeters range.

Another possible solution for self-localization that doesn’t use imaging is the employment of

radio frequency identification (RFID) in a 3D environment [36]. The robot is equipped with a reader antenna. At every robot motion, the distance between the tag and the antenna varies along the path. An equation utilizes the relative distances that were known at two different time instants, making it possible to know the exact locations of the robot. Experiments were conducted with simulated and real-world datasets to predict 3D and 2D localization, resulting in mean accuracy values as low as 13cm for 3D localization and 0.21cm for 2D localization. Tests also indicated that combining synthetic apertures on every 3D coordinate improved the method's estimation, obtaining errors below the centimeter for 3D localization.

A different method for self-localization of robots is to deploy visual landmarks in the environment. These can be artificial landmarks placed by researchers, and the robot then recognizes these utilizing a camera for capturing the image and an algorithm to identify them, and the information obtained is utilized to estimate the robot's position and orientation.

In [41], experiments were run in an indoor environment that resulted in reliable and accurate results, where landmarks, which in this work are circular symbols that each have their associated absolute world position, are placed on the ceiling of the environment. The robot possesses a camera that is mounted above it to visually detect the landmarks, and so achieve reliable and robust localization.

A combination of both artificial and natural landmarks can also prove to be a valid option for robot self-location, as seen in [15]. In this project, a combination of node-based topology localization and metric localization was implemented that used visual landmark detection to analyze the correct location node. A feature matching algorithm was used for topological localization with the natural landmarks, such as corners of walls, doors, etc. These can be insufficient for the algorithm to predict localization correctly, so experiments were conducted that tested the practical use of artificial landmarks in conjunction with the natural ones. These artificial landmarks are placed in dubious positions, where the robot has difficulty detecting the necessary landmarks to achieve localization. In previous metric localization methods, the robot would use a triangulation method that uses the positional relations among more than three landmarks to achieve localization. To avoid this, in this project, artificial landmarks added to the environment were constructed to help achieve localization with only one artificial landmark in view, which proved successful. This added dimension of landmarks and localization methods seems to complement each other to obtain a robust self-localization.

In 2021 [20], an image-based solution was proposed that utilizes a smartphone camera for the location of a person in an indoor environment, like an airport or train station. It proposes a solution that utilizes deep belief networks to classify different scenes and a perspective-n-point algorithm to localize the camera with several reference points extracted from depth images. The results obtained on this project have an accuracy up to the decimeters.

In [23], Long et al. present an accurate object detection method in remote sensing images by relying on CNN architectures. The data set was composed of aerial view images collected from Google Earth and Tianditu, and the framework of the project follows a pipeline that can be

divided in three processes: region proposal, classification, and accurate object localization. Region proposal uses a selective search algorithm that delineates areas over the original image where objects were detected, producing candidate object regions. In the next phase, a combination of two CNN models, AlexNet and GoogleNet, is used to extract features from these candidate features and classify them into four categories: aircraft, oil tank, overpass, and playground. To accurately localize the objects, they proposed a USB-BBR algorithm that augments the object localization precision and NMS to decrease the overlapped areas. This project resulted in a robust and simple solution that obtains optimal bounding boxes around objects.

In [5], Brubaker et al. present a map-based probabilistic visual model that achieves self-localization odometry calculated from two video cameras mounted on a vehicle and road maps. They used an approximate inference algorithm to compute the localization in real-time. The probabilistic nature of the model helped in avoiding the effects of noise in the visual odometry and the uncertainties that may arise in the maps. This method achieved a 4-meter self-localization accuracy over less than a minute of driving around 2km.

In [14], an approach is discussed for the improvement of visual place recognition by using a multi-sensor fusion method that, instead of relying on multiple physical sensors, it combines various image processing techniques on a single image stream and dynamically adjusts the sequence matching length based on recognition quality. Since there are multiple image processing techniques, only one needs to perform well in any environment which lowers the reliance on a single image processing method. It was evaluated on several challenging datasets, showing a great performance compared to state-of-the-art systems, achieving an average F1 score of 0.95.

Radwan et al. proposed in 2018 [29] VLocNet++, a multi-task learning framework for visual localization, semantic segmentation, and odometry estimation, that together enhance each other's performance. A strategy was implemented that encodes geometric and structural constraints into the "pose regression network by temporally aggregating learned motion-specific information and adaptively fusing semantic features". This was achieved by implementing an adaptive weighted fusion layer that learns better weights for fusion based on region activations. It was also developed a self-supervised warping technique that aggregates scene-level context in semantic segmentation networks that improving the performance of the model and decreasing the training time, while not being computationally demanding. In addition to this framework, a large-scale outdoor localization dataset was constructed, which contains loops and pixel-level semantic labels for the type of network that was developed in this research. The performance of the model was evaluated with Microsoft's 7-Scenes dataset and when employing the single task version achieved a translation error of 2.3cm, and 96.4% of prediction errors below 5cm, an error of 1.3cm, and 99.2% below 5 cm when employing multitasking. When compared with the then state-of-the-art Brachman's model, which had an accuracy of 76.1%, the proposed model surpassed it by over 20%. The multitask iteration of VLocNet++ also improved on the pose rotation accuracy by 25.9% (degrees of orientation error went from 1.04 to 0.77).

Constructing datasets for specific situations in image-based machine learning localization is

the focus of several studies, since the specificity of the type of input dataset helps ensure the positive performance of the model. Spera et al. [34] propose a large-scale dataset composed of 19531 RGB images captured inside a retail store to understand customer behaviour. The images were captured using a camera rig mounted on a shopping cart that moved along the environment. For every RGB picture is associated a depth picture captured at the same time, the ground truth position of the camera, and class labels that reflect areas of the store. The experiments were conducted with state-of-the-art algorithms, such as a pretrained VGG16 and Inception V3 architectures. For the classification task, it was obtained an accuracy of 96.4% when using RGB and depth images as input, which is an improvement when compared with when using only RGB images as input (95.1%). Regarding localization based on the camera pose estimation, the best mean position error was obtained with a single-task model,with a error of 0.42m, but the best mean orientation error was obtained with a multitask model, which obtained a 1.35°error. The multitask model that was trained for classification and camera pose estimation performed worse, with a 0.66 median position error and 2.38 mean position error. An interesting aspect to point out is that the models showed difficulty in locating certain zones correctly that were too similar visually.

In [35], a photorealistic dataset of a 3D indoor scene called Replica is presented as a solution to a 3D dataset that is primarily used for machine learning. This type of dataset enables studies that rely on the perception of the environment and can be compared to this project, as a more faithful representation of a room. It also has the same objective of trying to be applied to a possible real-life scenario present in this thesis. The difference between the two works is how the datasets are made. In the work described, it is obtained through a RGB-D rig with an IR projector to collect data that is used in a SLAM system and other algorithms that build the 3D environment, while in this one the environment is built on *Blender* utilizing it´s Python coding capabilities to program the design of each scene, which may be time-consuming but a more accessible method.

The creation of this type of 3D dataset with an intended use in machine learning is an interest in this area, as seen in [7], where a large dataset containing labeled images of an expansive indoor home environment is presented. It comprises of 194,400 RGB-D images obtained from camera rigs mounted on tripods placed in the environment that take 18 HDR pictures in a panorama. This project aims to provide a quality dataset with 360°panoramic RGB-D views along with semantic segmentation of the precisely replicated 3D world.

Another project utilized an image-based CNN for geo-location while taking into account the orientation of the images [22]. These images are spherical 360°x180°panoramas, capturing the view of the camera of around half a sphere. The learning model used is a Siamese CNN that receives different inputs in each of its two branches. One received the spherical panorama images while the other received a geo-referenced satellite dataset, and the model´s objective is to accurately feature-match the panorama image to the matched satellite image. An element studied in this paper is the application of orientation information for localization, which greatly simplifies the localization task when considering the geo-referenced dataset that has the North orientation labeled on the images. To this end, color-coded orientation maps are associated with each image, both

ground-view and aerial view, for feature embedding. This added dimension resulted in teaching the Siamese network geometric orientation and improving localization by 25% when comparing the performance when using orientation maps(93.19%) and not(68.61%), outperforming the then state-of-the-art CVM-net (91.54%) when comparing top 1% recall.

A recent work studied the application of the Internet of Things (IoT) in localization machine learning in indoor environments [30]. For this, the focus of the project was reviewing other projects that utilize wireless devices to help in localization inside indoor environments, in particular Received Signal Strength Indicator (RSSI), which are signal strength measurements from other access points or beacons. This data can be applied to ML to localize the position of the users and other IoT wireless devices. RSSI is a simple and cheap method to develop but has the drawbacks of limited accuracy and high variability. They conclude that this combination of methods hasn't achieved its potential and deserves to be studied further to understand its capabilities and limitations.

In [25], image-based localization is achieved utilizing an hourglass-shaped CNN network to predict orientation and location from a single RGB image. This hourglass structure is obtained from an encoder-decoder architecture that adopts the pre-trained CNN ResNet34 as an encoder, and a set of three up-convolutional layers for the decoder, to restore important fine-grained information from the input images that may be lost during encoding. At the end of the architecture, there is a regressor module that consists of three fully connected layers, each for localization, orientation, and translation. A skip connection was implemented to better preserve finer details by being placed between the residual blocks of the encoder to the corresponding up-convolutional layers of the decoder. The dataset used for evaluation was Microsoft's 7 Scene dataset, which contains images of 7 indoor environments. The proposed model achieved great results, with translation rotation errors of 0.24m, 10.24°(without skips) and 0.23m, 9.53°(with skips), surpassing state-of-the-art performance of models such as PoseNet (0.44m, 10.4°), LSTM-Pose (0.31m, 9.85°), and VidLoc (0.25m, N/A). This approach showcases an interesting take on self-localization with this different architecture from the VGG one used.

From the works mentioned, it is possible to observe that most of these studies used datasets obtained from real-world scenarios, indoors and outdoors. These are very valid options since their use in the context is easily proven with the results obtained, but it can be difficult to train robots in these settings. This difficulty gave prominence to projects that utilize artificial replicas of the real-world environment. So, one of the main focuses of this project is to prove the viability of the training learning model with artificial data, for possible use in real-world localization scenarios, utilizing easily obtainable datasets of said 3D artificial environments.

2.3 Dataset Quality

As described before, one of the topics explored in this project is the data dependency problem present in machine learning. This section will describe works that illustrate and present strategies to improve and guarantee the quality of the datasets to guarantee a positive performance of the model.

The performance of every machine learning model is largely impacted by the quality of the dataset used as input. Several studies focus on this data dependency problem to discover the best qualities the dataset has to possess to improve the model.

Different aspects of the problem of data importance in machine learning are explored in several papers that are analysed in the survey [13].

This article reviews these works and sets quality assertion processes that must be followed to ensure quality. The evaluation framework consists of 8 different quality dimensions, each a necessary quality to the dataset, and 32 evaluation metrics that gauge them. It also displays several limitations in measuring quality, such as the absence of uniformity in the evaluation metrics used in data gathering.

Different projects apply a distinct set of dataset quality dimensions, but try to comply to similar aspects that describe a widely accepted definition of quality. According to [16], the following set of 8 general dimensions was defined, each encompassing a group of more specific ones, providing a basis to ensure dataset quality, which this project will follow to achieve quality:

- Completeness – Related to the expectation that a value is attributed to each attribute, with no missing piece of data. If NULL values are correctly placed and clearly distinguishable from real values. The dataset should encompass and clearly represent every aspect of the task at hand, with the correct amount of data needed, and be relevant to the project.

- Availability and Accessibility – Related to how easily accessible the data is in its totality, whenever it is needed. This access should be controlled to avoid damage to the data and unauthorized access;

- Currency – Related to the timeliness of the data, it should be up to date with the objective of the work being conducted and should be able to be used on time to avoid using outdated information.

- Accuracy – Related to how much the data accurately represents the world it is representing. It needs to agree with the source and reflect the real-world values.

- Validity – Related to how the data conforms to the business rules established. The collection and representation of the data must follow the rules set by the business.

- Reliability and Credibility – Related to the trustworthiness of the data. The information must be unbiased and be obtained from trusted and credible sources, and must its lineage must be traceable.
- Consistency – Related to the consistency of the data, allowing for comparisons with different sets of it. The data must be identifiable, recorded in one place, and have the same semantics that describe it. Another very important aspect is that the dataset must not include conflicting information or be heterogeneous.
- Usability and Interpretability – Related to how usable and understandable the information is. The data must be relevant to the task it is being applied to, understandable, interpretable, and evaluated continuously through the organizational context.

In [28], it is suggested that this set of quality dimensions doesn't cover completely the many facets that involve machine learning procedures. The authors describe how in different projects it is vital to take into consideration a considerable number of dimensions are are solely unique to that problem. They illustrate this problem with a railway quandary that involves visually discerning lights through machine learning, concluding with defining data specification and verification, to achieve a more specified quality insurance on specific machine learning problems.

Further procedures can be made to maintain a higher level of quality in the dataset. In [26], measures are taken to mitigate the existence of noisy and mislabeled data, resorting to Confident Learning. This is a procedure that focuses on the matters stated previously, pruning noise from the dataset, calculating probabilistic thresholds that measure said noise, and ranking them with confidence for training use. This project detected labeling problems in the well-regarded *MNIST* dataset, which is a database of thousands of images of handwritten digits commonly used in image processing systems.

The quality of a dataset is determined by the aspects presented in this chapter, which are going to be present in the solution explored in this project. How these are maintained is explored in section 3.1.

Chapter 3

Data Generation

As mentioned previously, an important aspect of machine learning is the dataset used as input, how to obtain it, and ensuring the quality of the information it contains. The solution developed consists of the adaptation of the 3D modeling program *Blender*'s *Python* scripting capabilities that allow the creation of the dataset intended for training a model capable of localization.

This chapter contains four sections that describe different parts of the dataset generation process.

3.1 Using Blender to Generate Data

In this project, the main objective is to develop an *ML* model capable of localization in a virtual environment, using solely images as input, rendered in said virtual environment. This section explains how these are reliably obtained while maintaining quality.

The 3D modeling program *Blender* provides necessary features that enable the rendering of images that capture 3D environments. By using this program, several aspects referred in section 2.3 are easily achieved, avoiding very common problems associated with machine learning.

Obtaining the quantity of data needed is one of the main complications associated with machine learning. *Blender* trivializes this process since it renders any number of necessary images.

In addition to obtaining the dataset, another big issue that is present in *Supervised Learning* projects is the need for labeled data. The labels need to provide information about the data to the model in training. In real-world scenarios, consistently obtaining these labels while avoiding missing information and ensuring the truthfulness of said information is a difficult and expensive process for researchers. This is another advantage to resorting to artificially created datasets, where the information can be easily obtained with confidence that the information is reliable and correctly associated with each image.

Another feature that justified this choice was the incorporated *Python* environment in *Blender* that allows the creation of scripts that create and manipulate the environment, enabling simultaneous rendering and learning.

These advantages allow us to achieve the qualities described in section 2.3, which evaluate the quality of the dataset. How they are achieved is explained as follows:

- Completeness - The dataset and its labels are generated simultaneously, and if there aren't any errors in the code, *Blender*'s API guarantees that the labels are correct. The datasets also encompass the entire environment, due to *Sobol Sequencing* the rendering locations, as explained later in section 3.4.
- Availability and Accessibility - Since the dataset is created in this project, obtaining the needed quantity of data isn't a problem.
- Currency - The dataset is created at every experiment, meaning the images rendered are up to date with the current version of the environment being used at the time.
- Accuracy - The images are accurate to the scene by being rendered directly from the environment in the same program it is modeled in. The use of *Sobol Sequencing* in the distribution of rendering points greatly improves the representation of the scene in the datasets.
- Validity - Since we model the representation of the data, it follows the rules established for this project.
- Reliability and Credibility - We create the data, so the generated data follows the established settings and can be trusted.
- Consistency - the types of labels and their representation are consistent within the scope of the experiments, allowing comparisons. Conflicting information is a point of contention in certain settings explored in chapter 5. Still, it surges from the scenes established and isn't an inherited problem from *Blender*.
- Usability and Interpretability - Information is created specifically for this project, so the data is formed to be usable in the task at hand.

3.2 Blender API Overview

As mentioned briefly in the chapter 1, this project started with the overhaul of Cruz's work [12] to be better suited to accomplish image-gathering and create the datasets needed, as well as to enable a more general way to create image datasets. The objective of that project was to build a *Python* API for *Blender* that leverages artificial technologies and environments for the advancement of machine learning. This was achieved by creating methods that enable the generation of virtual worlds in the *Blender* environment, using a procedurally generated city as an example to illustrate the capabilities of the work done. The success of this thesis helps to solidify the quality of Cruz's project.

In that work, an object-oriented approach was taken where object classes were defined so that classes could represent the real-world entities present in the scenarios that could be modeled in *Blender*. It also enables inheritance between classes, an important factor when considering the need to model various objects that have different aspects but share basic functions and qualities, such as the different objects placed in a scene. This approach also promotes scalability and flexibility, which were the main objectives of Cruz's work when trying to create a procedurally generated artificial world.

For our project, this approach is maintained, and a lot of code is utilized, which greatly favors the quality and speed of the work being done. *Python* classes were maintained and expanded on to better suit the objectives set. The ones relevant to the modeling of the scenes from which the dataset will be formed are those related to the *Blender* objects that will be placed to construct the environment.

In *Blender*, the basis for every object is a 3D structure called a mesh. A mesh is a collection of vertices, edges, and faces that are organized in a way to form the surface of an object. A mesh *Python* class was developed as the basis for any object class that will extend from it. These extended classes will have access to the functions present in the mesh class, which were designed to manipulate any *Blender* object.

The first type of objects used were simple ones that could represent simple placeholder items that could represent real-world ones that may possess more complex designs. These consisted in spheres, cubes, cones, and tubes, all extended mesh *Python* classes. These followed the regulations reported below:

- Cube - a 3D mesh in the shape of a rectangular cube, where each pair of opposite faces has the same size and are parallel. When constructed, it's given a size for height, width, and depth. To take the shape of a cube, the 3 measurements need to have the same value.

- Cone - a 3D mesh that can take the shape of a cone or a cylinder. When constructed, it's given a radius for the top base, a radius for the bottom base, and a cone depth value that

translates to the height of the object. The shape of the object depends on the given size for the radius of the bases.

- Cylinder - a 3D mesh in the shape of a cylinder. When constructed, it's given a radius for both bases and the object's length. This mesh is a subclass of Cone, where the radius of both bases is the same value.

- Tube - a 3D mesh that takes the shape of a tube. When constructed, it's given a radius that goes from the center to the outer side of the object, a thickness value for the tube, and the length. This mesh is a subclass of Cylinder, where the area of a cylinder of radius equal to the difference between the radius and the thickness is removed from the bigger cylinder with the given radius, utilizing the difference operator of the Mesh class.

From these simpler objects, more complex ones can be built utilizing functions from the mesh class that can join and delete areas of objects with others.

Shelf is a *Python* class that describes a geometrically simple shelf where items can be distributed in its row, allowing for a better representation of how smaller objects could be stored. This object is formed from the manipulation of the size of several cube objects that form the rows and spine of the structure, which are afterwards joined together in a single object to form the shelf. It also possesses a function that allows the user to place objects in three designated locations, along each of the three rows, allowing for easier arrangement.

The mesh class isn't limited to the simple forms defined in code, but can also import *Blender* objects from STL files. An STL (*Standard Tessellation Language*) file characterizes 3D geometric surfaces commonly utilized in 3D modeling and printing programs.

This functionality allows for more detailed and complex structures and objects to be displayed in the environment. This kind of object makes the scene more realistic but more computationally demanding, which can cause problems in the scene-building phase and image-rendering process.

STL files are easily accessible online, with several free and even more premium alternatives. Because of all this, the building process of a realistic scene with the means to form an image dataset is easily achievable for users with minimal *Blender* 3D environmental knowledge.

Since *Blender* employs a dynamic lighting system, light sources are needed for it to be able to observe anything when rendering images. Light objects are a different type of *Blender* object that must be present in the environment to achieve this. These simulate real-world lighting, which brings depth, realism, and mood to a scene, since they define how surfaces reflect, cast shadows, and interact with their environment. Their absence would cause the objects in the scene to be flat or even completely obscured.

A *Python* class was needed to manipulate through code for such an important feature of the program. Some functions of this class include the resize of the radius of light cast, and one that allows the focus of the light rays to a certain object (only applicable to certain light subtypes).

Subtypes of light exist that possess distinct characteristics that influence the environment differently, to better replicate distinct forms of casting light. A *Python* class was created for each of them, which extends from the light *Python* class. They are the following:

- Pointlight - light emitted from a single point in space. It's used to project light at a very specific point.
- Sunlight - light that tries to emulate the sun, emitting parallel rays of light of the same intensity.
- Spotlight - light emitted in the form of a cone that can focus on a specific point.
- Arealight - light emitted in from a square area of variable size.

In the figure 3.1 is a class diagram that easily conveys the relations between classes and some of their more important functions.

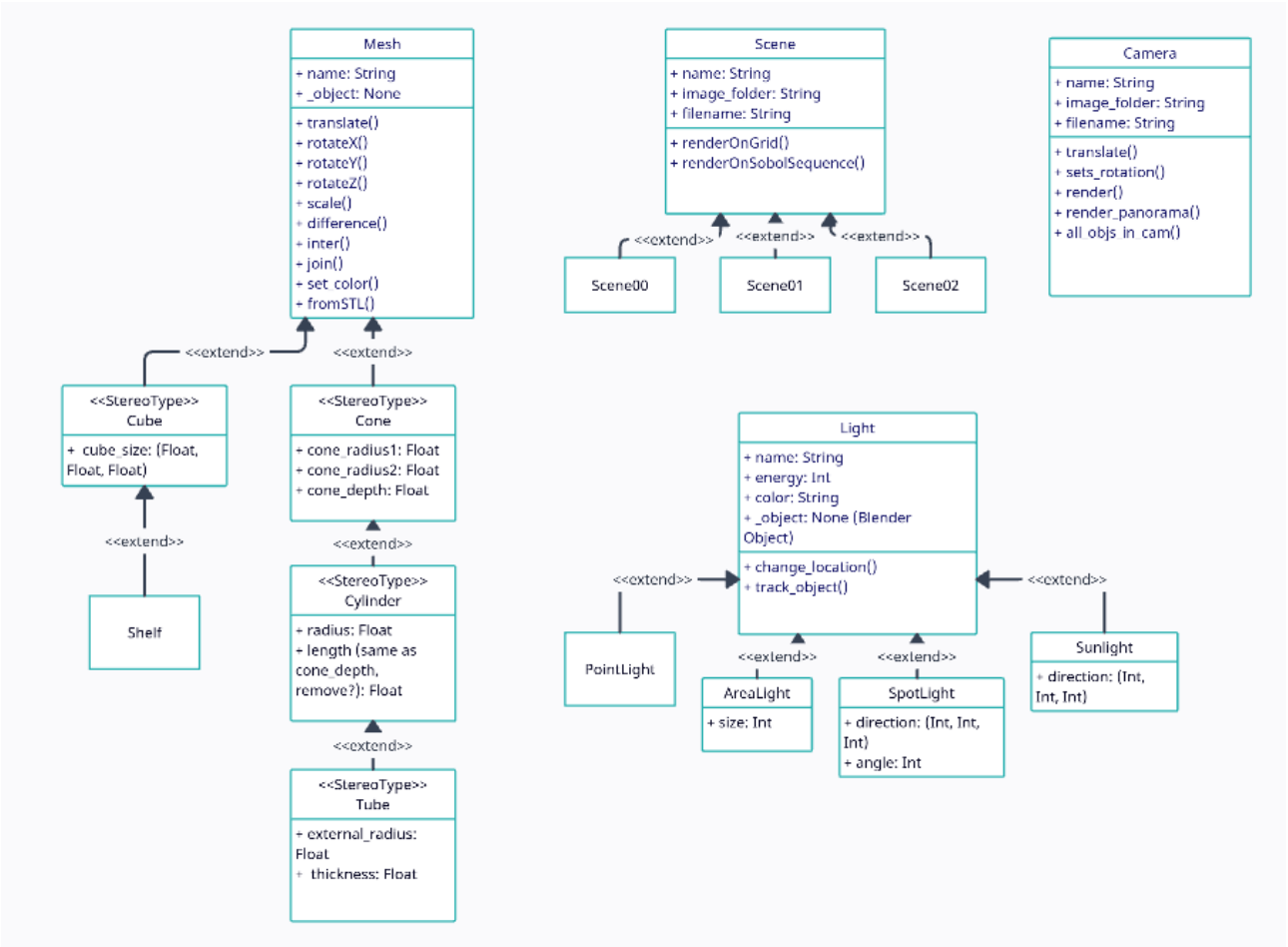


Figure 3.1: Class diagram representing the relations between Python classes, that translate to Blender objects.

3.3 Scenes Design

Scenes were created to generate datasets, where images will be rendered and utilized as data. They are formed by organizing the objects described in the previous subsection, emulating scenarios that could be described as a simple warehouse. For each consecutive scene, they become bigger, more detailed, and more complex. This allows for great progression on how realistic the scene becomes, and how the dataset can replicate cases closer to real-world scenarios. One aspect that is important to mention is that for a truly realistic replica of real-world scenarios, it is very important to use measurement units and scales that correctly reflect the real environment; however, for this project, *Blender*'s measurement units are considered conceptual. From this point on, any mention of dimensional and distance measurements is in these conceptual measurement units.

A simple scene called *Scene00* was created as a test scene where the first mechanics of the project were put to test and developed. It is a simple 200x200x100 closed space, with a floor, ceiling, and four walls. Four objects, each of a different *Python* class, are placed relatively equidistant from each other in each quadrant of the environment. They also have different colors to better distinguish between them, and to help in the learning process of the Neural Network. Light sources are placed in the ceiling to illuminate the environment and make the objects visible to the camera. There are four of them, and all are Pointlights, to better emulate the light cast from light bulbs. In the image 3.2, we can observe this scene.

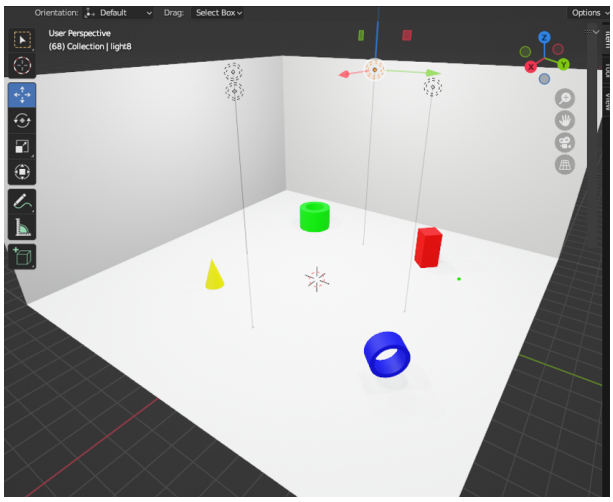


Figure 3.2: Representation of *Scene00*. The ceiling and front walls were removed for better visualization.

The first datasets were obtained from this scene, which were used in the first batch of training and testing of the model. Several variants of this scene were also developed to conduct experiments that further explore the learning process. These will be presented and discussed in chapter 5.

The next scene is labeled *Scene01*, and creates a larger and more complex world with dimensions of 700x700x300, that aims to challenge the model's robustness and its capacity to adapt to

more structured environments. It is represented in the figure 3.3.

It can be characterized mainly by its ”crossroads” of rows of shelves that add a simple but defined set of structures, creating a uniformly constructed scene. A shelf consists of a structure with three racks where objects can be placed in specific locations, three in each of these rows.

In this scene, objects of differing shapes, sizes, and colors are placed along these shelves, only one allowed per rack, for brevity's sake. These objects change the model's perception of the environment, where each row in itself looks different from its counterparts because of these objects.

As described, this leaves big zones where images gathered wouldn't possess relevant information about the environment, which would be a deterrent to the project. So, shelves with objects that follow the same rules dictated previously are also present, placed on the opposite side of each shelf.

Lights are distributed in the ceiling, as such is a vital component in the scene that allows for dynamic representation of real lighting and for the camera to capture the images in its view. Different from the previous installment, these are *SpotLights*, which create a more focused light source, which is represented by a cone of light. These allow for the lights to be focused on the more important features of the world.

The set of elements described all contribute to a higher degree of complexity, which, with the added layers of uncertainty, may cause noise in the gathered data. These factors will demand more from the model and require a thorough evaluation of its pattern-recognizing capabilities.

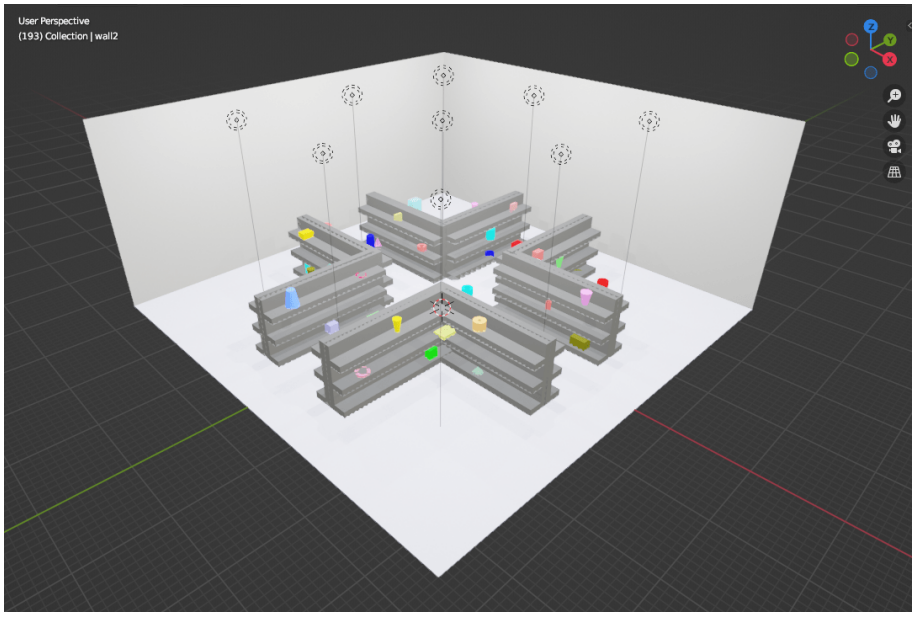


Figure 3.3: Representation of *Scene01*. The ceiling and front walls were removed for better visualization.

3.4 Dataset Rendering

The Camera object type in *Blender* is responsible for the visualization of the environment. It can render images of what it captures in its view, following the set parameters.

A Camera *Python* class was also created to develop different camera manipulation procedures that enable more customization of the object through code and also allow specific image-capturing methods.

The image rendered is the same as the visualization from the camera's point of view. In the *Blender* view, a projection of this vision is present, which makes the camera's orientation and the objects in view clear.

An analysis was conducted to determine the FOV(Field Of View) of the camera and the resolution of the image to obtain a good representation of the scene that captures useful information for the machine learning model. A set of 16080 images was rendered, and the metric used to evaluate it was the number of objects in view. Besides the FOV, only the width changes instead of both resolution dimensions since the objects are placed at the same z level, so more vertical information isn't needed in this scene.

16080 images	Average Number of Objects in View		
Resolution	**FOV = 36 (DEFAULT)**	**FOV = 90**	**FOV = 120**
128x128	0.57	1.14	1.47
256x128	0.54	1.13	1.47
512x128	0.50	1.10	1.45

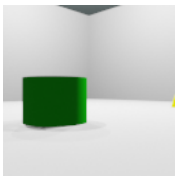
Table 3.1: Results from the number of objects analysis of a dataset with 16080 images

From the results obtained in the table 3.1, it is clear that maintaining a square resolution results in more objects being present in the image. Altering one of the rendering dimensions results in an image that can be interpreted as a "cropped" version of the original view, but with more pixels representing its width. The images 3.4 demonstrate this outcome.

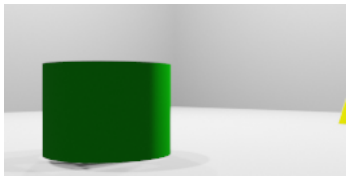
However, altering the FOV is much more impactful on the number of objects in view than changing the resolution. This increase of FOV captures more of the scene but greatly distorts the image, which may impact the performance of the model. The images 3.5 compare the FOV values used in this analysis.

The image rendered with the FOV of 90 captures a lot of the environment, but the objects appear very distant to the camera and alter the perception of the dimensions of the scene. The one rendered with FOV of 120 displays high distortion, to the point that the vision intercepts the walls and results in the dark grey bars on the sides.

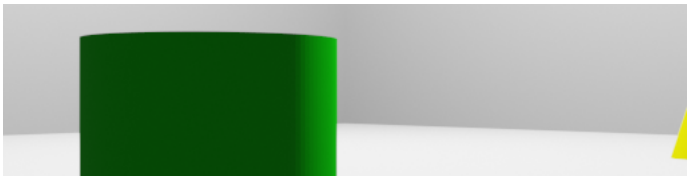
The resolution of the images in the datasets to be used in the experiments was decided to be 128x128, which is a smaller resolution that allows a less computationally demanding process



(a) 128x128



(b) 256x128



(c) 512x128

Figure 3.4: Comparison between images with different resolution

to generate and be processed by the model, while keeping a level of detail that can perceive the scene, especially one as simple as *Scene00*. The FOV stays the default 36°value of *Blender*, which renders an image with a faithful representation of the scene with no distortion.

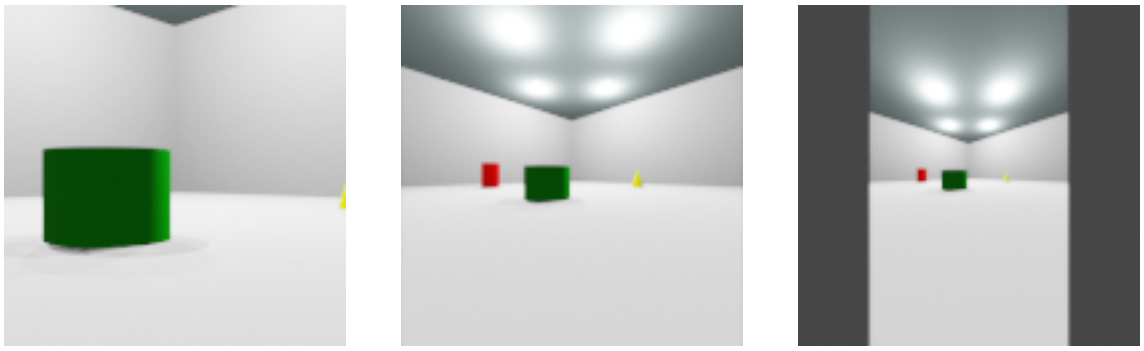
Now that the rendered images have been established, the methods used to generate the datasets are going to be described. The first rendition of the dataset generation method is called *renderOnGrid()*. It consists of the positioning of the camera at sequential points of a horizontal 2D grid, parallel to the xy surface.

The user defines the number of cells, the size of the grid, and the height(z coordinate). At each point, the camera will face the horizon, with the x rotation value at 90. Then it is rotated in the z-axis until it reaches the original orientation, allowing for a panoramic view of the point where the camera is situated. This rotation is defined by a step value that determines how much the camera rotates to render a single image. The number of images rendered at each point is 360/*step*.

A rendition of this grid placed over *Scene00* is demonstrated in the picture 3.7, where at every crossover between lines of the grid, images are rendered. The dimensions of the grid are 8x8, which corresponds to a 25 measurement units range of precision, since *Scene00* has a 200x200 floor dimension, and 200/8=25. As explained in section 3.3, these units of measurement are conceptual and should not be directly compared to real-world dimensions.

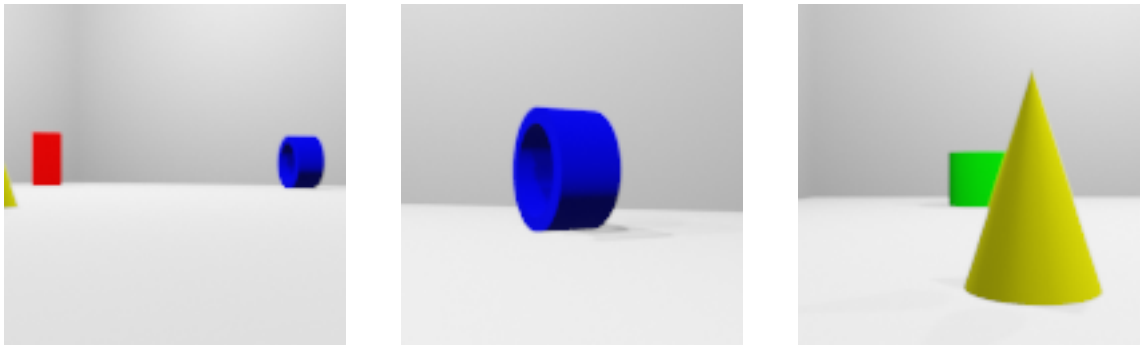
A problem was encountered where images would still be rendered when a rendering point was inside an object, so a functionality was added where if the point coincides with the volume of an object, images are not rendered, and that location point is skipped.

The structured form of how the images are obtained also limits how to use this method in dif-



(a) FOV = 36 (Blender Default) (b) FOV = 90 (c) FOV = 120

Figure 3.5: Comparison between images with different FOV (Field Of View)



(a) FOV = 36 (Blender Default) (b) FOV = 90 (c) FOV = 120

Figure 3.6: Examples of images captured in *Scene00* that form the datasets used in the experiments

ferent environments, is not very flexible when considering large-scale environments, and doesn't generate images in point of varying Z coordinates. The problem related to the intersection of the camera position and the volume of objects also causes the obtained images to be lower than the expected amount.

A new image rendering method was formulated to fix the issues related to before. To avoid the limited grid structure of the previous method, a new one was proposed based on the Sobol sequence.

The *Sobol Sequence* is a low-discrepancy sequence used in quasi-random number generation, which helps ensure that points are distributed more uniformly across space than traditional random number sequences. The sequence of locations resulting from this method is represented by the black spheres distributed in the *Scene00* in the image 3.8.

This procedure allows for a more varied dataset that closely resembles the real scene. The sequence is applied to the three coordinates, which expands from the 2D variability of the past method to the third dimension. The limits for each coordinate are defined when running the function and should have values inside the room built in the environment.

Which sequence point is used to render the images is chosen randomly. If one of these points is inside an object, this point is skipped, and another one is chosen at random until the obtained point is outside the volume of the item. This is done to avoid the problem of when images are

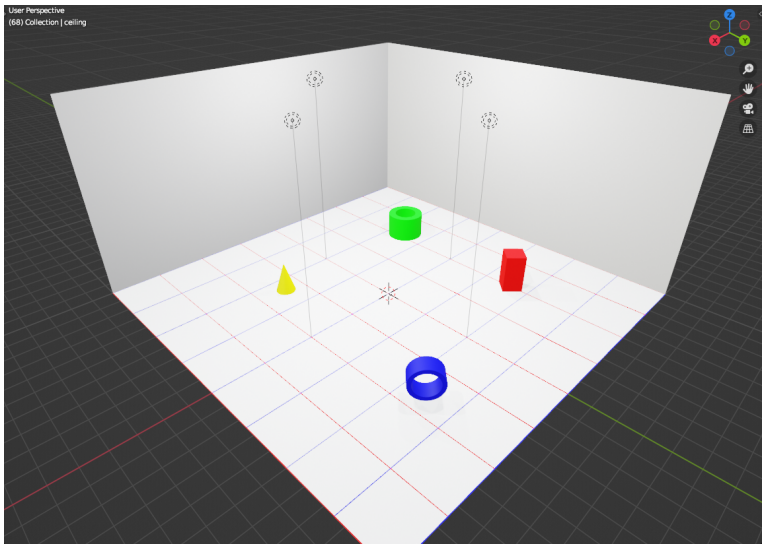


Figure 3.7: Representation of a 8x8 grid over a *Scene00* environment.

rendered inside objects, which means that images would not possess information about the scene. In the previous version, the skip would happen, but that point would be substituted for another, and images would be lost. The new solution provided allows for a constant number of images to be rendered.

A problem still present in this version of the function is that images that don't include items still don't provide any useful information to the training model, being considered noise. A fast solution to this is to only allow for images that include items.

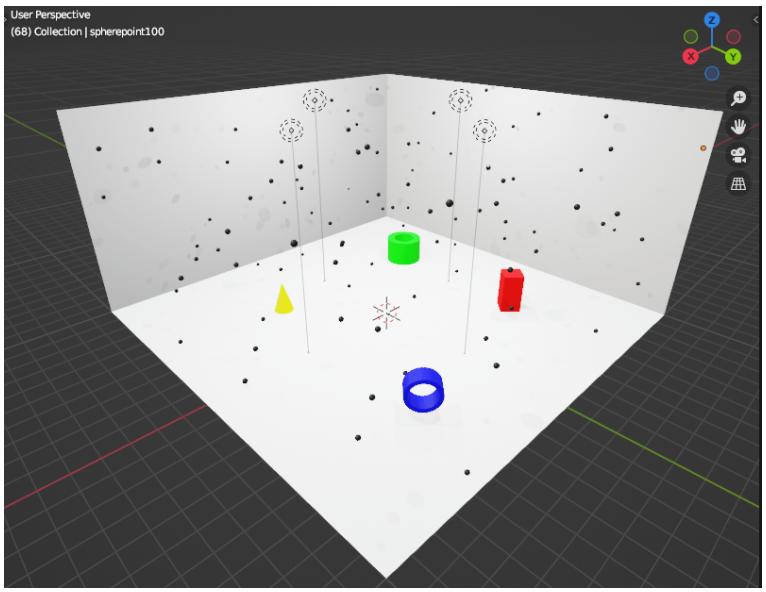


Figure 3.8: Representation of the locations generated following a generated *Sobol Sequence*. Each black sphere represents one of the possible rendering locations. *Scene00* used as an exemplary environment.

Three types of object detection functions were added to the camera *Python* class to detect

every *Blender* object in its view. The techniques detect objects using ray casting to detect different aspects of the items in the *Blender* environment. These aspects are: the vertices, the center of a face of the object, and the origin point. Considering all these features allows for a more thorough detection with minimal errors.

A list of objects of interest is defined, and if none of them are detected, then that image isn't rendered, skipping to the next angle step.

Experiments were conducted using the techniques described in this section, but another one was developed to avoid the use of image filtering while trying to deliver a solution to the problem of featureless images that are damaging to the model's performance. The problem is when the camera renders images that are directed at a wall, which only causes noise. Experiments that expose this problem are described in chapter 5.

This last solution aims to give information to these images that don't contain objects in their view. It consisted of changing the previous *Sobol Sequence* method by pairing the images with a front and back view, allowing for when the camera is facing a wall without information, its pair will retain information about the items on its opposite side. This kind of data aims to solve this problem better, as it recontextualizes the images to fit in the overall environment, facilitating the localization process. These experiments are explained in section 5.2.

In every single one of the previously explained dataset-gathering procedures, as an image is being saved, a JSON is created to retain information about it. This is done to calculate loss evaluations in the training process and calculate the quality of the model, which consists of a supervised training process. The data saved is the image name, coordinates, and rotation values of the camera when the image is rendered and the objects in its view. The structure of the JSON file is the following:

```
{
    "image_name": "image-005",
    "coordinates": [
        90.55,
        60.52,
        2.7
    ],
    "rotation_values": [
        90.0,
        0.0,
        90.0
    ],
    "objects_in_view": [
        "platform",
        "q1",
        "t2",
        "wall1",
        "wall4"
    ]
}
```

This information is solely used when evaluating loss and accuracy. The model only receives

the generated images as input without any context.

Chapter 4

Machine Learning Model

The *Artificial Neural Network*, more commonly used for image-based learning, is the Convolutional Neural Network (*CNN*), since it can solve pattern recognition problems in images and assign importance to the features it encounters [27][21]. A lot of computational power and processing time is needed to train a *NN* with an image dataset, but *CNN* was built to be able to process this type of input, thanks in part to the same set of weights shared between different parts of the input, reducing the number of parameters.

The architecture of a *CNN* consists of these layer types:

- **Input Layer** - the first layer that receives the raw data needed for training. Needed in every *ANN*;
- **Convolutional Layers** - set of learnable filters that perform convolution operations over the input data. These operations involve element-wise multiplication of the filter and a region on the input image, and then these products are summed together. A feature map is obtained from the application of this operation over every possible position across the input image. After this, an activation function like *Rectified Linear Unit* (*ReLU*) is used to apply an element-wise operation that allows only the output of positive values, introducing non-linearity to the network;
- **Pooling Layers** - downsampling is executed along the input, reducing the feature map, and so, reducing the number of parameters while still maintaining the relevant information and removing noise;
- **Fully-connected Layers** - tries to determine an output from the sum of weights of the features obtained from previous layers. In the case of classification tasks, the results are passed through another activation function, obtaining the probability scores needed for classification.
- **Output Layer** - the prediction/classification is provided from the features learned throughout the learning process. Needed in every ANN.

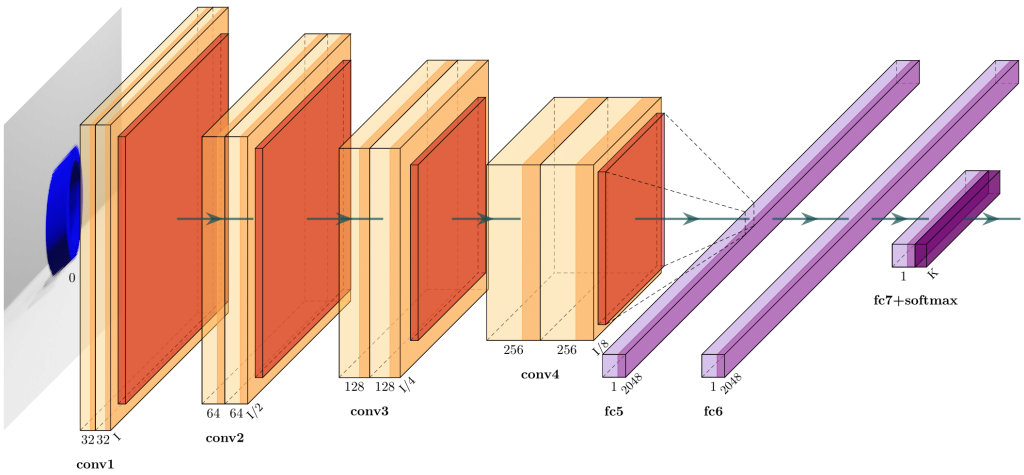


Figure 4.1: Representation of the *Convolutional Neural Network* architecture used for the learning model in this project.

A visual representation of the architecture that describes the models used in this project can be seen in the figure 4.1, and its structure is described as follows:

```
{
{'miranda': [[128, 128, 3],
   ["conv2d", 32, 3, 3], ["relu"],
   ["conv2d", 32, 3, 3], ["relu"],
   ["maxPool", 2, 2, 2, 2],
   ["conv2d", 64, 3, 3], ["relu"],
   ["conv2d", 64, 3, 3], ["relu"],
   ["maxPool", 2, 2, 2, 2],
   ["conv2d", 128, 3, 3], ["relu"],
   ["conv2d", 128, 3, 3], ["relu"],
   ["maxPool", 2, 2, 2, 2],
   ["conv2d", 256, 3, 3], ["relu"],
   ["conv2d", 256, 3, 3], ["relu"],
   ["maxPool", 16, 16, 16, 16],
   ["fc", 2048], ["relu"],
   ["fc", 2048], ["relu"],
   ["fc", 11]]}
}
```

For starters, the model receives input images with dimensions 128 by 128 pixels, with 3 RGB color channels. These images have a relatively small resolution, but their size allows for the creation of a larger dataset while maintaining a lower level of computational power needed to generate them in the dataset-gathering phase and process them in the learning phase, while still conserving the quality of the image and not compromising its visibility of the scene.

Afterwards, there is a set of four convolutional blocks. Each block comprises two consecutive convolutional layers, each of them with 3x3 filters, and is followed by a *ReLU* activation layer. At the end of the block, a maxPool layer is placed that halves the dimensions of the input, except in

the last one, where the input is 1/16 of the previous one, having a dimension of 1 by 1.

This downsampling of the input greatly reduced the size of the feature maps, resulting in fewer parameters and weights in the next layers and reducing computational power cost, simplifying calculations made in the training phase. Feature maps being represented in minimal dimensions create a global overarching summary of the encounter features, representing the general context of the input image. The model learning from more generalized data also helps to avoid overfitting, forcing the model to learn from only the most important information in each image. Still, this solution may have some drawbacks, such as the removal of relevant information when downsampling.

The architecture ends with two *Fully Connected* layers that classify the input from the feature maps, resulting in probabilities for the output. Each of the layers is followed by a *ReLU* activation function. After these, an output layer with 11 units is set, but isn't definitive. This value varies at every experiment to fit the specific dimensions of its labels.

This structure is based on the VGG architecture, which is a type of CNN that was previously explained in detail in section 2.1. One of the main topics described was the depth of the network and its importance, and how this aspect is used to designate a name to the model. Considering the 4 sets of 2 convolutional layers, followed by 3 fully-connected layers at the end, the total number of learning layers is 11, meaning the architecture explored could be considered a VGG-11.

The machine learning algorithms explored in this project were built by employing TensorFlow[1], which is a highly scalable open-source interface created by Google engineers that allows the development of algorithms, including the deep convolutional neural network model intended in this project. The project benefits from this library thanks to its compatibility with *Python*, which is the programming language implemented in *Blender*'s scripting. A drawback that surges with this choice is the incompatibility between certain versions of Tensorflow and its dependencies, including certain Nvidia libraries needed to run models utilizing a GPU, where if one of these got an update, it would cause TensorFlow not to work as intended. To avoid this issue, the TensorFlow code had to be rewritten to use the PyTorch library, which proved to be much more stable.

This model structure will be applied from now on to every experiment. It will be applied specifically in two different localization solutions: one that uses classification to identify areas that illustrate the robot's location and another that uses regression to measure distances between the real and predicted values.

These two categories of experiments are further categorized based on the labels used to identify each picture, the techniques used to obtain the images, and the scene in which the experiments are being conducted. The following sub-chapters describe these tests in more detail.

4.1 Classification

In the classification experiments, the labels of each image correspond to three characteristics of each image: X coordinate, Y coordinate, and angle. These possess values that are not continuous, but instead identify zones defined in the environment.

A grid with the same number of cells on each side is delineated, and each cell formed is considered a location. The X and Y coordinates identify the cell coordinates; X is associated with the row, and Y is associated with the column. The label Angle identifies the orientation of the camera when the image was rendered, which helps the robot place where positions and objects are in relation to each other. This label also follows the same logic applied to the location coordinates, where the available scope of rotation of the camera is divided in sections with the same degrees, and each section is identified by a sequence of numerical values that increment at each new rotation zone. The resolution of the grid defines the number of angle zones. So, for example, if a 4x4 grid is established, that means a total of 12 classifications exist, 4 for X, 4 for Y and another 4 for the angle.

For each type of experiment, a different loss function was defined for the model, since classification and regression require functions that better suit the problem.

A *Softmax Cross-Entropy* loss function with 3 logits was chosen for the iteration of the model. This function combines *Softmax* and *Cross-Entropy* into a single efficient operation that avoids complications that might arise from utilizing them separately.

Logits are raw output generated by the model assembled at the final fully connected layer. They are commonly produced for use in classification operations.

The number of logits chosen is based on the number of labels in the dataset that are used for classification. Since those are the X coordinate, Y coordinate, and angle value, 3 logits were the logical choice. The function utilized is a *Softmax Cross-entropy with 3 Logits* loss function.

The Softmax function converts logits into probabilities, normalizing them into numerical values between 0 and 1, where their sum results in 1. These values represent the probability distributions for each classification class. The following is the formula for this operation:

Softmax formula

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

where:

- z_i: Logit for class i,
- N: Total number of classes.

After the softmax probabilities are obtained, the cross-entropy loss is calculated to measure the accuracy of the predicted probability distributions, if they correspond to the true distributions.

The resulting loss value indicates if the model is correctly progressing toward the goal of correctly predicting the classes. The following is the formula for this operation:

Cross-Entropy formula

$$\text{Loss} = -\sum_{i=1}^{N} y_i \log(p_i)$$

where:

- y_i: True label for class i (1 if correct class, 0 otherwise),
- p_i: Predicted probability for class i from the softmax function.

The loss function utilized in this iteration of the model is a combination of both of these, resulting in a single operation that receives the logits directly and outputs the cross-entropy loss. This streamlines the process, resulting in a more stable and efficient method. The formula of this combined function is the following:

Softmax Cross-Entropy with Logits formula

$$\text{Loss} = -\sum_{i=1}^{N} y_i \log\left(\frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}\right)$$

where:

- z_i: Logit for class i,
- y_i: True label for class i (1 if correct class, 0 otherwise),
- N: Total number of classes.

4.2 Regression

In this phase, several regression solutions will be explored that share the same model architecture as the classification version, differing in the labels that describe information about each image in the dataset. An aspect shared in every iteration is that the labels' values are continuous, allowing for a more precise representation of the real and predicted location.

Mean Square Error (MSE) is the loss function chosen for this iteration of the model. This is commonly utilized in regression models, measuring the average squared difference between the predicted and the real target values.

Squaring both values ensures that all values are positive, in cases where the difference is positive or negative, both having the same weight as each other. Another advantage is that larger errors correspond to a bigger emphasis on this value, which weighs more than smaller differences. Below is the formula for this loss function:

Mean Square Error formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where:

- n: Number of data points,
- y_i: Real value for i,
- $\hat{y}_i$ Predicted value for i.

An additional model architecture was constructed that also features regression. In this new architecture, the model learns from a pair of images simultaneously to add further visual context to each label. This pair of images corresponds to opposing views, one a forward view and the other a backward view, with a 180-degree rotation between them. This duality emerged as a solution to when the robot would not capture objects and other relevant information in its camera view. In these cases, where the camera only captures a wall, for example, then the backward image would be able to capture objects and other valuable information situated opposite the wall.

To accommodate this added dimension, the model is bifurcated into two parallel learning paths that share weights between them and converge at the end by concatenating the outputs of the feature extraction layers. One of them is fitted to the front view and the other to the back view.

Chapter 5

Model Training and Results

The experiments that were conducted to achieve localization are described and explained in this chapter. The results obtained from experiments greatly shape the direction of the tests, allowing for a fluid progression between them that justifies each other.

At first, these experiments are simple to serve as a baseline that should be surpassed at every iteration. The conditions in which they occur differ to understand the cases that greatly improve the model's performance, along with studying the influence of the environment, the method of generating the data set, and the learning procedures on the model.

The first training experiment is simple in its representation of location and in the environment from which the images are rendered. As this chapter progresses, the scenes change to add more complexities to the environment to understand its influence on the performance and determine the possible viability of this project in real-world applications.

5.1 Scene00 Grid Classification Experiments

This section describes and explains the first experiments, which explore a supervised machine learning classification model that uses a square grid, forming a very simple localization that is being used as a ”proof of concept” of this project.

The algorithm tries to predict the location of the robot in the *Scene00* environment with a class-based representation of location that matches a grid that divides the scene into location areas. The coordinates of the location are the classes predicted by the model, one class for X, and another for Y. The combination of these two coordinate classes determines the cell of the grid that describes the location zone.

The technique used to capture the images is the *Sobol Sequencing* described previously in which location points inside a predetermined area that follow the *Sobol Sequence* are calculated and selected randomly as image rendering positions, where the camera will render images around itself at certain degrees of rotation. In these experiments, the degrees of rotation are 18°, creating 20 images in a full 360° rotation.

A simple experiment was conducted that evaluates the performance of the grid classification model in the simplest scene, *Scene00*. The resolution of the grid is 3x3, meaning there are a total of 9 possible locations and 3 classes for X and 3 for Y. How the location labels translate into the X and Y classes is described in image 5.1.

The Z coordinate isn't a relevant localization element to predict since the placement of objects is always constant in this coordinate, only possibly changing to specific height levels if stored in a shelf, which in a real-world scenario would be pre-established constant heights. This coordinate was chosen to be ignored, given the low relevancy and the added dimension of difficulty that would be predicting this coordinate.

The learning procedure, which will be the standard for the next tests, is a fold-based learning. This fold approach helps mitigate overfitting by changing the input dataset and allows for dataset generation and model training to occur simultaneously. A fold is an isolated dataset that is generated for the specific test, meaning no two folds contain the same images. At each learning process, a single fold is used as input at a time, meaning for a determined number of epochs, in this case 100, the model will learn from that fold. At the end of the 100 epochs, a new fold is used as input, and the model trains for another 100 epochs. The dataset used for validation is the one rendered after the current one, for example, for the first fold, the validation dataset corresponds to the second fold. This process repeats for the number of folds that were established, which in these experiments is 5. At the end, another data set with the same number of images as a single fold is formed to serve as a test set that is used to evaluate the performance of the model. In this case, the number of images per fold is set to 8000.

Following the settings defined in the last paragraph, the results obtained are presented in the table 5.1. The metric used for evaluation is accuracy, which is the percentage of correctly predicted

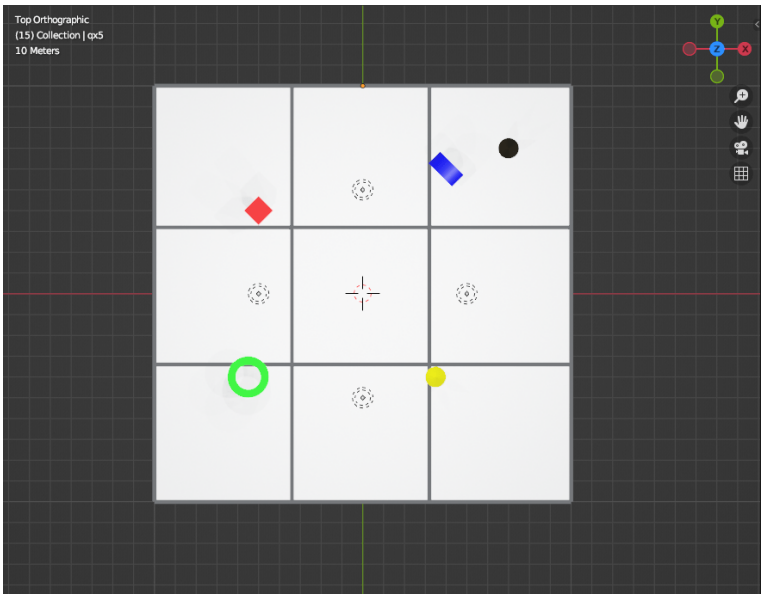


Figure 5.1: Representation of a rendering point in *Scene00*, with a grid on the floor that translates to the 3 possible X and Y classes; the rendering point is represented by a black dot in the environment, with coordinates of (70,70); this location is then translated into two X and Y classes, X=1 because it's in the first row, and Y=3 because it's in the third column.

locations, that is, both X and Y classes.

3x3 grid classification	
Images per Fold	**Accuracy**
8000	56.73%

Table 5.1: Results of the first grid classification experiment

The results obtained are very negative, with an accuracy of 56.73%, even with a very reduced precision and a low number of classes to predict. The main reason for this bad performance comes from the data used, because the method used gathers a lot of useless information since the scene in question contains a very limited quantity of objects of interest and a lot of blank space. Another problem that arises from these images with little to no visually distinguishable information is that images that are essentially the same visually have labels with different information. Cases where this happens are images rendered close to walls and corners. Examples of images with similar visual information but very different labels are presented in the figure 5.2. If this occurs in large quantities in a way that overshadows the ones with the objects of interest in view, it is harmful to the model's performance.

To overcome this issue, another experiment was conducted where an object filter was developed that, at the rendering stage, determines if at least one object of interest (that isn't a wall or floor in this instance) is in camera view and only renders if positive. This method allows for the datasets created to only possess useful information. The experiment follows the same settings as

(a) location of (-48.26, 29.82, 3.72) and rotation of (90.0, 0.0, 108.0)

(b) location of (46.30, -14.61, 2.55) and rotation of (90.0, 0.0, 198.0)

Figure 5.2: Images with very similar visual information but with very different labels

the previous one, and its results are demonstrated in table 5.2.

3x3 grid classification	
Images per Fold	**Accuracy**
8000	94.18%

Table 5.2: Results of the second grid classification experiment with the object filter

The case in which the object filter was activated resulted in the best performance, which is to be expected in this case, with a great accuracy of 94.18% compared to the previous 56.73%. Even if the application of this filter isn't a very thorough representation of the scene, since the parts with the least interest are removed, it results in better model performance, so for the next couple of experiments, this filter will be applied until a different method is used later on.

With the dataset settings now defined, more experiments that change based on dataset volume and grid resolution were conducted to test the limits of this approach. The table 5.3 demonstrates the results obtained in each case, where the learning process occurs along 5 folds, and the results are the accuracy of the model in said settings.

Images per Fold	**3x3**	**4x4**	**8x8**	**16x16**	**32x32**
1000	85.50%	78.00%	59.40%	38.50%	15.8%
2000	89.90%	86.00%	74.15%	54.40%	27.30%
8000	93.51%	92.31%	84.53%	71.64%	50.69%

Table 5.3: Grid Experiments results

The results obtained demonstrate that the volume of the dataset greatly improves the performance of the model, with the one with the greatest accuracy being the case where the resolution is 3x3 and the dataset a size of 8000 images per fold (a total of 40000 images). Even though the result is very positive, this doesn't translate to good precision.

As the grid resolution increases, in other words, the precision of the model increases, the accuracy dramatically decreases, demonstrating the non-viability of this classification approach. Nevertheless, this technique provides proof of concept that the next solutions will surpass.

5.2 Scene00 Regression Experiments

The experiments conducted in this section use regression as their learning method for localization, differing from the previous approach. This modification significantly improves the precision of the previous iteration, as it is no longer limited to the number of possible classes that previously defined the coordinates. *Scene00* is the chosen environment to obtain the dataset images from, meaning the environment contains very little detail, and so also does the dataset rendered. The advantages and disadvantages that come from this kind of experiment are also explored in this chapter and compared with the previous iteration.

This iteration of the model is called *XY3AF*, and it changes not only the model's output but also the representation of the information in the image, which will be explained in this section.

As mentioned in the previous chapter, this rendition of the model applies regression, and so a new loss function (*Mean Square Error*), along with a different loss measurement. This change permits a continuous representation of the error, allowing a precise distance between the real and predicted locations.

With this change, it is needed to change the representation of the data that informs about the image. Previously, these would simply be whole numbers (integers) that would classify a specific coordinate, which is a very limiting technique, so in this new one these are the real coordinates where the images are taken in the environment, represented by a float, which also allows for a decimal and more precise location coordinates.

The angle of camera rotation of when the image is rendered is also an aspect to be predicted in this iteration, and it's represented by 3 values. One is the degrees of rotation on the Z axis, which is a certain α degrees, the second is α+120°, and the third is α+240°. These values are specified to avoid discontinuity around 0°, where images rendered at these low rotations closely resemble ones rendered at values close to 360°. So, if α is vastly different between images, but the other set of values are close to each other, then that means that the images are being rendered in very similar angles.

The further specification to localize the angle at the same time as the position gives more characteristics to the images, which gives more information about the environment that helps the model to more easily understand the scene.

The dataset generation keeps the *Sobol Sequencing* technique, with the addition of information about position and angle following the newly established rules. For the first experiment, the object filter isn't present to define a starting point for this new iteration of the model. The filter will be applied later to compare the results obtained.

The result of this first benchmark experiment is shown in 5.4.

The results showcase that this method alone is not sufficient to achieve localization, so a set of experiments that apply the object filter that only allows images that capture objects to be rendered and added to the dataset.

XY3AF-nofilter	
Parameters	Average Distance
5 folds 4000images/fold 100 epochs/fold	50.41

Table 5.4: *XY3AF-nofilter* experiment result

The experiments were conducted into three separate groups that each focused on one specific parameter, dataset size, number of folds, and learning epochs per fold. By altering these values, it allows a better understanding of how these settings alter the potential of the model and the limits of the current implementation. Each of the tables 5.5 to 5.7 groups the tests for each of these parameters and presents the average distance between the predicted values and the real ones. Certain parameters need to stay consistent within each group of experiments to evaluate the specific parameter at hand, and are defined in each table.

Fold Size Experiment - XY3AF 5 folds, 100 epochs/fold	
Images per Fold	**Average Distance**
500	70.70
1000	30.23
2000	15.65
4000	7.90
8000	3.99

Table 5.5: Results of the fold size *XY3AF* experiment in *Scene00*

Number of Folds Experiment - XY3AF 4000 images/fold, 100 epochs/fold	
Num of Folds	**Average Distance**
3	11.36
5	7.90
10	5.25
20	4.97

Table 5.6: Results of the number of folds *XY3AF* experiment in *Scene00*

The results are positive in the *Scene00* environment, and a big improvement compared to the one without the object filter. The lowest average distance to the real values is 3.99, when the hyperparameters are 5 folds, 8000 images, and 100 iterations per fold. Considering this is a 200x200 environment, this error translates to an accuracy of 1.995% when compared to the length of the scene. The relative error was calculated as follows:

Iterations per Fold Experiment - XY3AF 5 folds, 4000 images/fold	
Num of iterations	**Average Distance**
50	10.95
100	7.90
200	5.91
400	5.58
800	5.61

Table 5.7: Results of the number of iterations per fold *XY3AF* experiment in *Scene00*

$$\text{Relative Error (\%)} = \frac{\text{Average Error}}{\text{Environment Side Length}} \times 100$$
$$= \frac{3.99}{200} \times 100 = 1.995\%$$

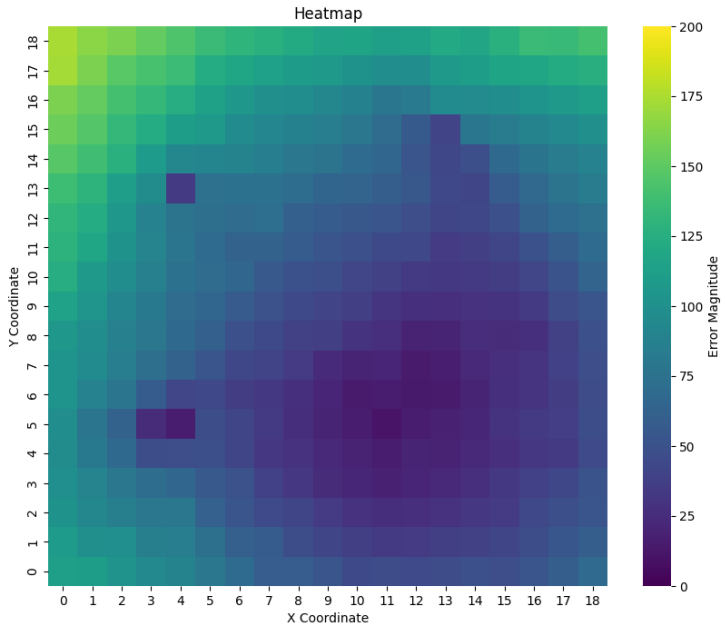
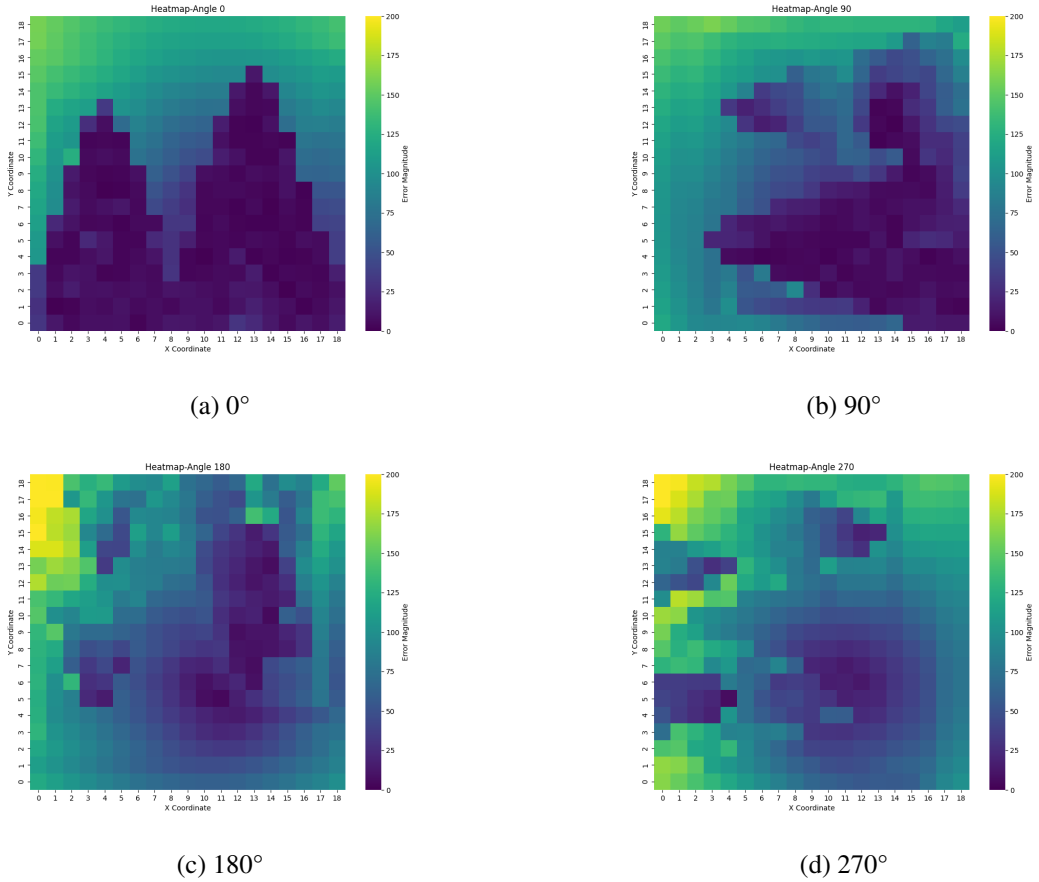


Figure 5.3: Average distance error heatmap of a *XY3AF* model with object filter, that can be interpreted as an above view of *Scene00*

While the object filter resulted in an improved performance, it still demonstrates a clear weakness in the current version of the model, which is how images rendered close to the walls are difficult to localize. This is an issue present from the previous iteration.

In figure 5.3, a heatmap is presented that represents the average error of predicted localizations in the *Scene00* environment, tested on the *XY3AF* model that applies object filtering. Each square



(a) 0°

(b) 90°

(c) 180°

(d) 270°

Figure 5.4: Average distance error heatmaps of a *XY3AF* model with object filter in *Scene00*, based on rotation, to visualize how the error varies

represents the average error calculated from averaging the cumulative error of the predicted localizations of all 20 images rendered in that spot, each with a different rendering angle. The difficulty of localization near the walls can be observed in the heatmap, where the locations closest to the wall showcase higher average distance error.

In the images 5.4, four heatmaps are presented, where the model was tested only on images rendered in the same locations as the general heatmap, but only images rendered in a specific angles. The angles used for each heatmap are 0°, 90°, 180°, and 270°, and the same conclusion can be arrived at, that in locations where objects of interest aren't captured, including closer to walls, the average distance error dramatically increases.

Images rendered in these areas not only have low to no information but also create noise because the walls are all the same gray color and don't possess any distinguishable features, resulting in what is visually the same image relating to a different point with different labels. To resolve this problem, a new iteration of the model was developed that changes how the model receives and interprets the images.

This new solution is named *XY3AF_stereo* and uses a dual-image representation of the environment as input, where the pair of images represents the front and back view of the camera. This

solution aims to solve the previously established problem by giving each image more visual and informational context by pairing it with its opposite. In this scenario, when the camera is positioned only to capture the wall (non-useful information), the image that it renders is associated with the one rendered 180 °later, which in most cases corresponds to a more informative image.

This new solution aims to substitute the object filter present in the previous iterations, where only images that capture objects of interest are rendered. With the new image pairing system, the full spectrum of images must be rendered at every point so that the airs can form, so excluding images without objects in their view isn't possible. In the same vein, the number of images in a dataset needs to be a multiple of the number of images rendered in a full rotation, which so far is 20 per point, with an image being rendered every 18°. To guarantee this happens, the datasets are multiples of a global constant called *BATCH_SIZE* of value 60. This constant also determines the number of images fed to the model at one time. That is why 60 was chosen instead of 20, because 20 would result in too little information at once.

Because of this detail, the exact number of images used in these experiments cannot be the same as in the previous iteration, so it is the closest possible number above them. The rest of the parameters are the same as the previous iteration, and the results are presented in tables 5.8 to 5.10.

Fold Size Experiment - XY3AF_stereo 5 folds, 100 epochs/fold	
Images per Fold	**Average Distance**
540	83.85
1020	72.64
2040	68.74
4020	66.30
8040	66.67

Table 5.8: Results of the fold size *XY3AF_stereo* experiment in *Scene00*

Number of Folds Experiment - XY3AF_stereo 4020 images/fold, 100 epochs/fold	
Num of Folds	**Average Distance**
3	67.09
5	66.30
10	67.89
20	66.98

Table 5.9: Results of the number of folds *XY3AF_stereo* experiment in *Scene00*

The results are disappointing, with a performance far worse than the previous version of the model. The increase in the number of images in the dataset, number of folds, or learning epochs doesn't seem to greatly impact the performance, with the best average distance between real and predicted values ranging between the same values of 66-67.

Iterations per Fold Experiment - XY3AF_stereo 5 folds, 4020 images/fold	
Num of iterations	**Average Distance**
50	66.96
100	66.30
200	66.81
400	67.82
800	68.68

Table 5.10: Results of the number of iterations per fold *XY3AF_stereo* experiment in *Scene00*

A further analysis was conducted to understand this discrepancy better. The graph image 5.5 is a comparison between the loss of these latest 3 versions of the model and the 3 experiments conducted under similar settings is presented.

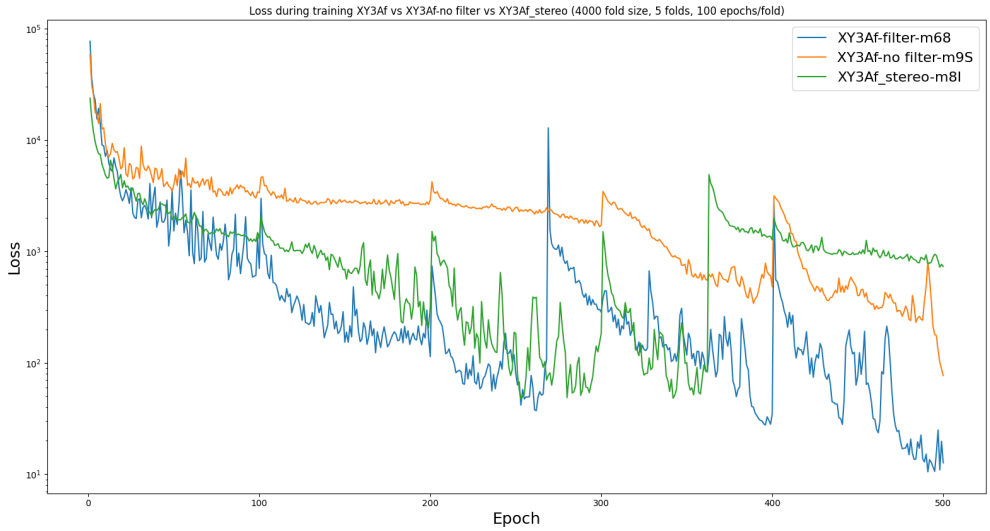


Figure 5.5: Loss graph comparing the three versions of *XY3AF*: *XY3AF*(no filter), *XY3AF*(filter), *XY3AF_stereo*

The *nofilter* experiment demonstrated a bad performance but still a relatively better one than *XY3AF_stereo*. This can also be perceived in the loss comparison graph 5.5, where the version with the largest loss is *XY3AF_stereo* due to several loss spikes that do not drop to significant levels during the learning process. These loss spikes occur with the changing of learning datasets when changing folds, along with the ones that occur in the gradient descent process. In the next section 5.3, a new learning procedure will be applied to try to fix this issue, abandoning the fold method.

One possible justification for the poor performance of *XY3AF_stereo* is the scene on which it is being tested on, which results in a lot of grey images with little to no visual information that muddles the images contained in the learning datasets, hurting the performance of the model.

Adding more objects and other distinguishable features may prove to be beneficial to the model. In sections 5.4 to 5.6, experiments that explore this hypothesis will also be explored, where the environment will undergo changes to reduce to a minimum the featureless images, and so improve the model's performance.

5.3 Batch-based learning process

The loss graph 5.5 in the previous section demonstrated that using the fold method results in loss spikes that are too damaging to the learning process, so it was decided to explore another learning technique. The new one consists of a big dataset before training that the model learns from throughout the whole training process. This avoids the constant loss spikes that occur when changing between folds during the learning process.

This method allows the addition of new rendered images to the existing pool of images that the model has access to at the beginning of the learning process. This new method aims to avoid the loss spikes associated with the change of folds, while still adding images along the learning process that helps to shorten the learning process and avoid overfitting. However, this technique wasn't applied to the experiments present in this thesis, to avoid an extra variable that isn't consistent between all the experiments. Future work that takes advantage of this feature is discussed in Chapter 6.

The images are rendered in batches following an established *BATCH_SIZE*. The value of this constant increases from the previous section to a value of 240. The validation and test datasets are generated before the start of training, with 30000 and 60000 images, respectively, which correspond to 125 and 250 image batches. The images of these datasets are rendered per the scene that the model is being trained on. These experiments were run with a long learning phase up to 10000 epochs and with a starting 2000 image batches(480000 total images).

To set a baseline and justify the use of the method, an experiment was conducted on *Scene00* with the *XY3AF* version of the model that only uses as input a single image, and the dataset generation doesn't use the object filter. The result was as follows:

Continuous Batch Learning 10000 epochs 2000 batches	
Model	**Average Distance**
XY3AF	34.35

Table 5.11: Results from *XY3AF* batch-based learning experiments without object filter in data generation in *Scene00*.

This experiment already showcases an improvement from the previous learning process, where the average distance was 50.41, which is a worse result than the 34.35 achieved. This being the case, the next experiments will use this new learning process.

To set the same standards used in the previous section, two additional experiments will be explored, one of *XY3AF* with the object filter and another with *XY3AF_stereo*. The following tables register the results of both experiments:

The object filter variation of *XY3AF* did not achieve good results with the learning batch technique, having a worse performance than the version without the filter. This shows how the filter

Continuous Batch Learning 10000 epochs 2000 batches	
Model	**Average Distance**
XY3AF (obj filter)	43.63
XY3AF_stereo	39.94

Table 5.12: Results from batch-based learning experiments in *Scene00*.

isn't a reliable solution to the problem and isn't inherently an improvement over the original one.

The stereo variation, while displaying a big improvement regarding the accuracy of the model, still falls short of single-image *XY3AF*. This could still be a matter of the scene in question not being compatible with this method, since details and distinguishable features should greatly improve its performance. This possibility will be explored in the next section 5.4.

For later comparisons with *Scene01* in section 5.7, the percentage of the average error compared to the length of the scene is 17.18% for *XY3AF* and 19.90% for *XY3AF_stereo*.

5.4 Scene00 colored walls experiments

This section presents experiments that explore improvements applied to the environment to better understand the limitations of the current version of the model, which should greatly benefit from said improvements. These improvements are coloring the walls of the scene in different colors to distinguish them better, since the featureless walls are a great detriment to the model's performance.

The addition of different colors to the walls almost creates a sensation where a different orientation is associated with a specific color, so it can easily tell both the walls and orientations apart and locate itself in relation to these elements.

These experiments will be conducted in a new version of *Scene00*, which will be aptly named *Scene00_colored* and will analyze the performance of both *XY3AF* and *XY3AF_stereo* models. These experiments will be conducted under the same specifications as the last section to help understand how beneficial this change is to each model and if it boosts their accuracy in different ways. The results are displayed in the table below.

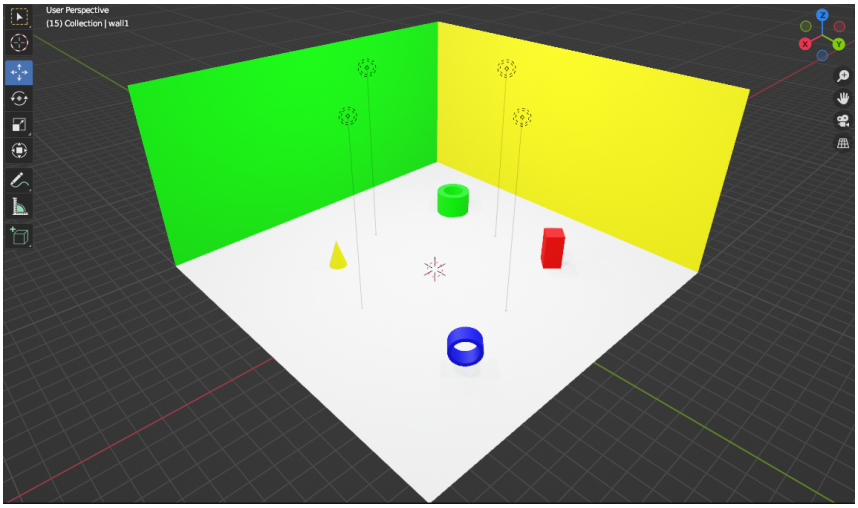


Figure 5.6: Representation of *Scene00_colored* environment

Continuous Batch Learning 10000 epochs 2000 batches	
Model	**Average Distance**
XY3AF	5.32
XY3AF_stereo	7.92

Table 5.13: Results from *Scene00_colored* experiments.

For comparison purposes later in the chapter, the percentage of the average error compared to the length of the scene for *XY3AF* is 2.66% and for *XY3AF_stereo* is 3.96%.

The performance greatly improved, proving three aspects: (1) the added color helped both models' performance; (2) the colors help distinguish the walls and better understand the environment; (3) the problem in the previous iterations that limited the performance were the featureless walls that created noise and conflicting information in the dataset.

In the previous section, an object filter was applied to the learning datasets as a way to mitigate the large number of featureless and colorless images, but it did not improve performance when using the learning batches method. Using color on the previously gray walls solves this problem, as seen by the results presented in the table 5.13.

However, this change was expected to benefit the stereo model further and achieve lower distance errors than the single input image variation. This not being the case, the *XY3AF_stereo* may be more limited than *XY3AF*.

More experiments regarding the colored walls were also conducted to understand the influence of the colored walls on the model's performance. This set of 3 experiments alters the saturation of the colors present in the walls with each installment, progressively closer to the original gray color from the original *Scene00*. Each installment is considered a new variation of *Scene00* and are called *Scene00_colors1*, *Scene00_colors2*, *Scene00_colors3*, becoming less saturated as the scenes go along. Examples of images from these scenes are presented in figs. 5.7 to 5.9, and the results of the experiments in table 5.14.

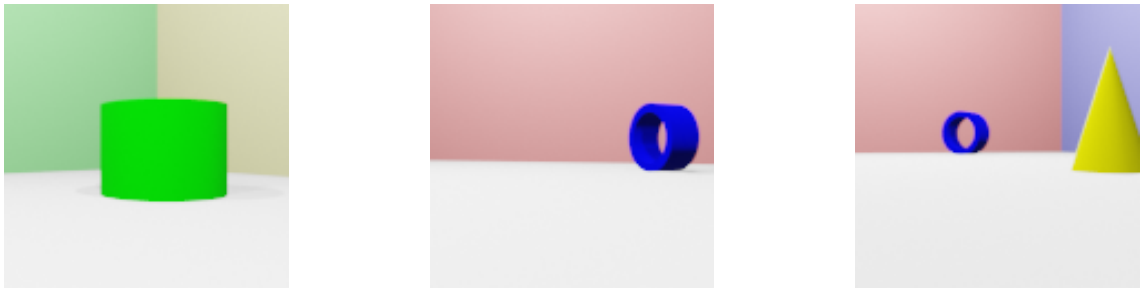


Figure 5.7: Examples of images rendered in *Scene00_colored1*

Figure 5.8: Examples of images rendered in *Scene00_colored2*

The results show that as the colors become less saturated, the walls are not as distinctive as before, and the model's performance slightly worsens. However, in the case of *Scene_colors3*,

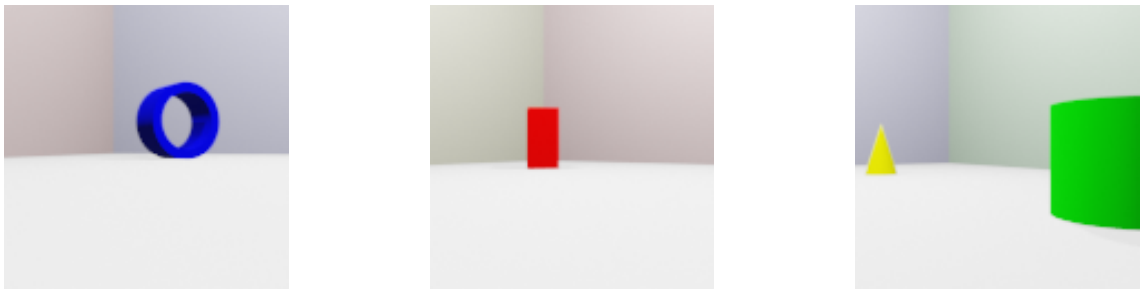


Figure 5.9: Examples of images rendered in *Scene00_colored3*

Continuous Batch Learning - Average Distance 10000 epochs 2000 batches		
Scene	**XY3AF**	**XY3AF_stereo**
Scene00_colors1	5.19	8.79
Scene00_colors2	5.44	8.05
Scene00_colors3	5.76	9.43

Table 5.14: Results from colored walls tending to gray experiments of *XY3AF* and *XY3AF_stereo* models in *Scene00_colors1*, *Scene00_colors2* and *Scene00_colors3*

where the rendered images closely resemble the original *Scene00* with only a slight hue of color in each wall, the performance greatly improves to 8.73 when compared to the results of 39.8 obtained in section 5.3. This revelation admits that a small change to a featureless/ambiguous element or zone of the environment can greatly improve localization. These experiments also continue to support the fact that *XY3AF* achieves a better performance than *XY3AF_stereo* when conducting experiments under the same conditions. Sections 5.5 and 5.6 explore these points further by adding elements to the scene that act as markers that help achieve localization.

5.5 Scene00 colored stripe on walls experiments

This section explores different interpretations of coloring the wall, where instead of coloring the entire wall, a colored stripe of variable size, depending on the experiment, is placed from the ground up. The length of the stripe covers the entire length of the wall, but the height is different depending on the scene, to test if a more limited change compared to painting the entire wall helps achieve a similar model's performance.

Three scenes were created to this end: *Scene00_stripe1*, *Scene00_stripe2*, and *Scene00_stripe3*. The height of the stripe is respectively 1/2, 1/4, and 1/8 of the wall's height. Examples of images rendered in these scenes are showcased in the set of images figs. 5.10 to 5.12. The experiments follow the same conditions as in the previous sections and are presented in Table 5.15.

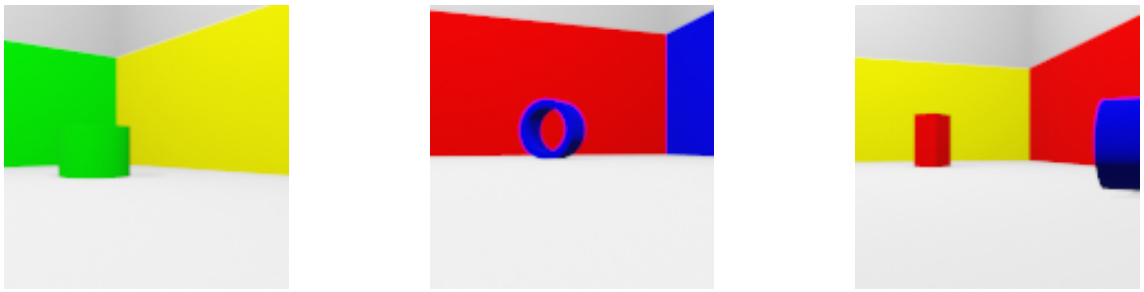


Figure 5.10: Examples of images rendered in *Scene00_stripe1*

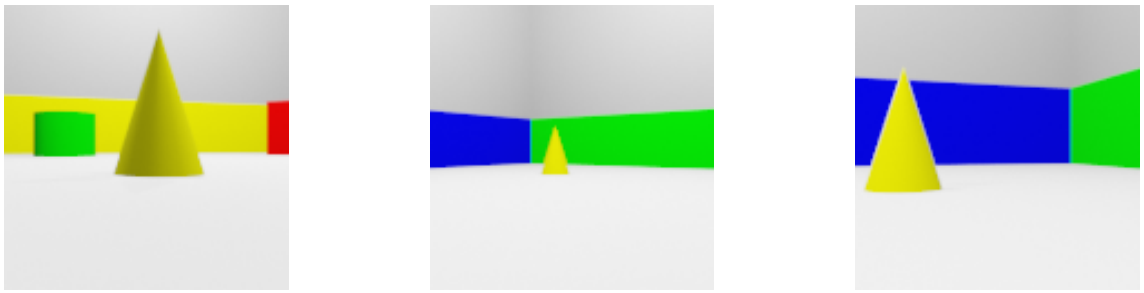


Figure 5.11: Examples of images rendered in *Scene00_stripe2*

As expected, the performance of the model increases as the area covered by the colored stripe increases. This is because with the increase in size of the visually distinct areas and less of the featureless grey wall is visible. However, the implementation of the smallest stripe still provides a significant improvement compared to the original *Scene00* with gray walls, as seen in Tables 5.11 and 5.12. This fact continues to support the fact that a small addition to visually distinguish areas that are otherwise featureless greatly improves localization accuracy.

Both version of the model achieved their greatest performance in *Scene00_stripe1*, surpassing

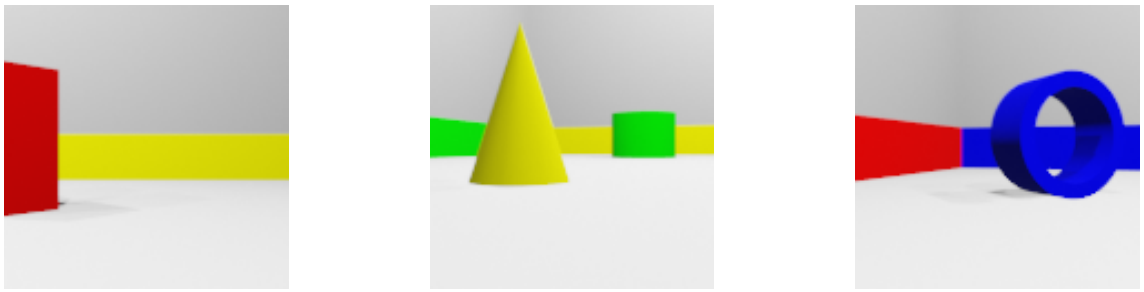


Figure 5.12: Examples of images rendered in *Scene00_stripe3*

Continuous Batch Learning - Average Distance 10000 epochs 2000 batches		
Scene	**XY3AF**	**XY3AF_stereo**
Scene00_stripe1	3.02	6.94
Scene00_stripe2	4.08	8.33
Scene00_stripe3	5.01	9.34

Table 5.15: Results from colored stripes on walls experiments of *XY3AF* and *XY3AF_stereo* models in *Scene00_stripe1*, *Scene00_stripe2* and *Scene00_stripe3*

the fully colored walls experiment. The colored stripe captured in the image may provide more information compared to the fully colored walls because of the angles formed between the colored and grey portions of the walls, which almost create a sense of orientation.

5.6 Adding Objects to Scene00

Visual markers may provide more visual information on the scene and help the model achieve localization more accurately. In this section, experiments will be described in which different types of elements will be progressively added to the environment, with the hope that accuracy will improve progressively. The specifications of the experiments will follow the ones used so far, using both *XY3AF* and *XY3AF_stereo* models to understand the advantage the single-input image has over the stereo two-input image version.

In the first variation, parallel lines are placed on the floor. This added element should help the model to situate itself in the scene, because the lines act as markers that can act as measurements of the scene's length. This scene is called *Scene00_line*s and a representation of it can be seen in image 5.13, and example images in 5.14.

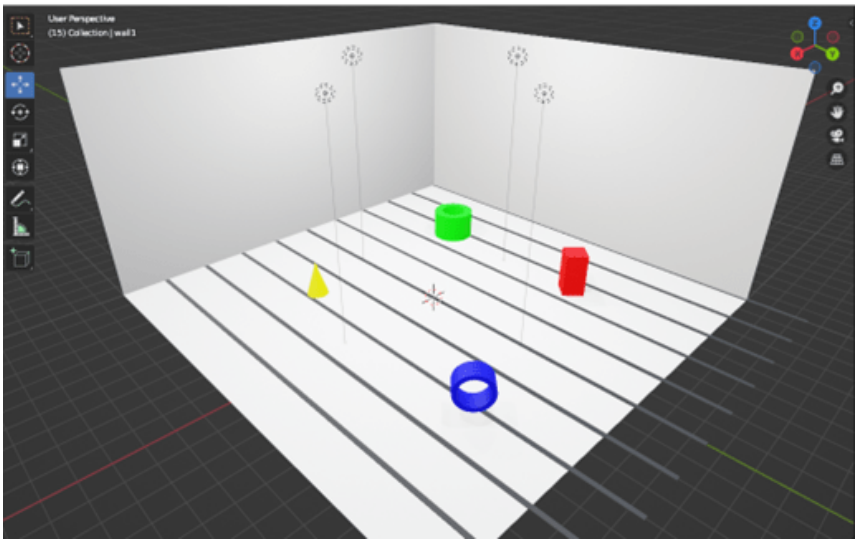


Figure 5.13: Representation of *Scene00_lines* environment

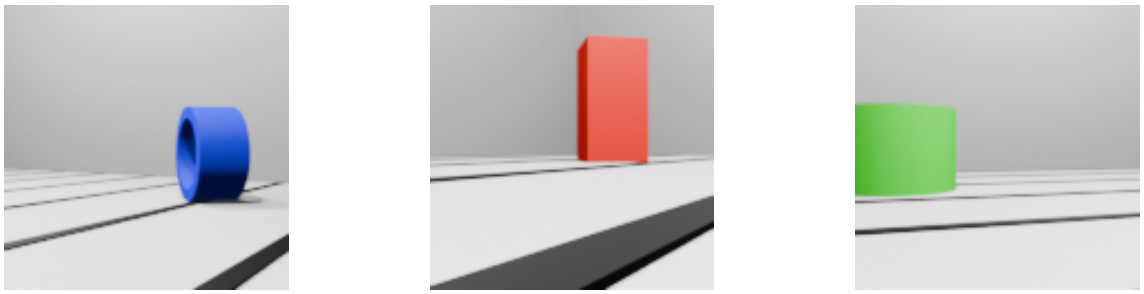


Figure 5.14: Examples of images rendered in *Scene00_lines*

In the next scene, a line grid is placed on the floor. Following the same logic in the previous

scene, creating this grid should help recognize locations. This scene is called *Scene00_grid*, and a representation of it can be seen in image 5.15, and example images in 5.16.

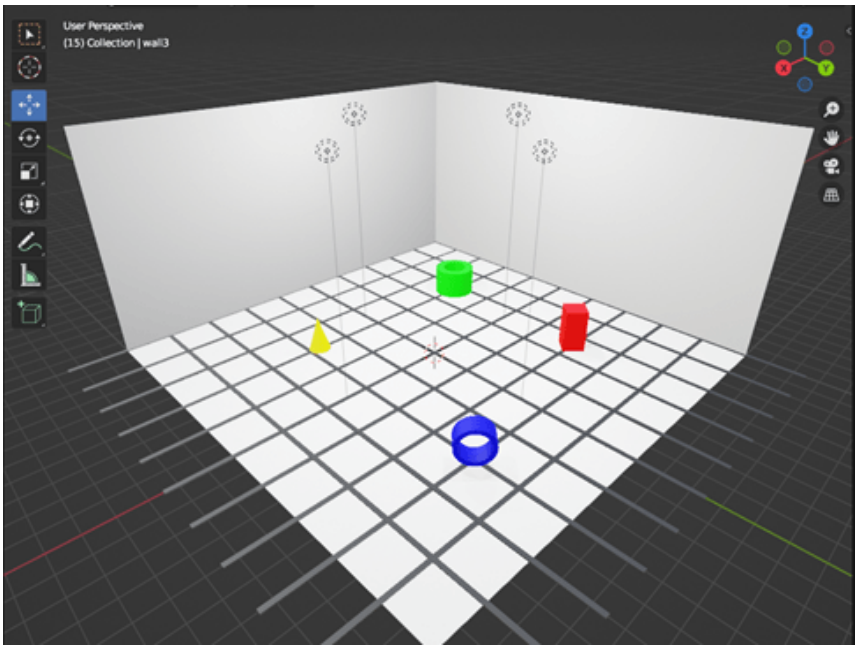


Figure 5.15: Representation of *Scene00_grid* environment

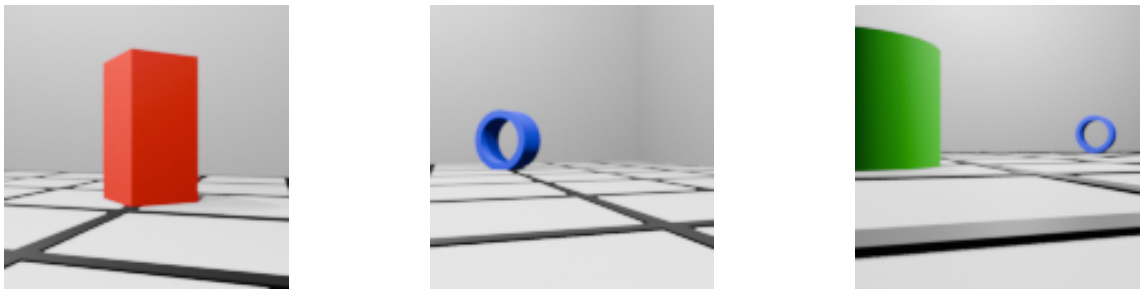


Figure 5.16: Examples of images rendered in *Scene00_grid*

Adding a different type of visual cue may help differentiate even better the images and so help localization. For this matter, three signs that show different letters were added to each wall. The signs are equidistant to each other, covering the wall so that the camera is more likely to perceive at least one of them when looking at a wall. This scene is called *Scene00_grid_signs*, and a representation of it can be seen in image 5.17, and example images in 5.18.

Another experiment in a scene called *Scene00_colored_grid_signs* will serve as the best possible augmentations from this set, by combining all the alterations applied so far, adding color walls to the previous iteration of *Scene00*. A representation of this scene can be seen in the image 5.19, and the example images in 5.20.

The results of the experiments described in this section are shown in 5.16.

Every single experiment confirmed that the presence of added elements, as simple as they may

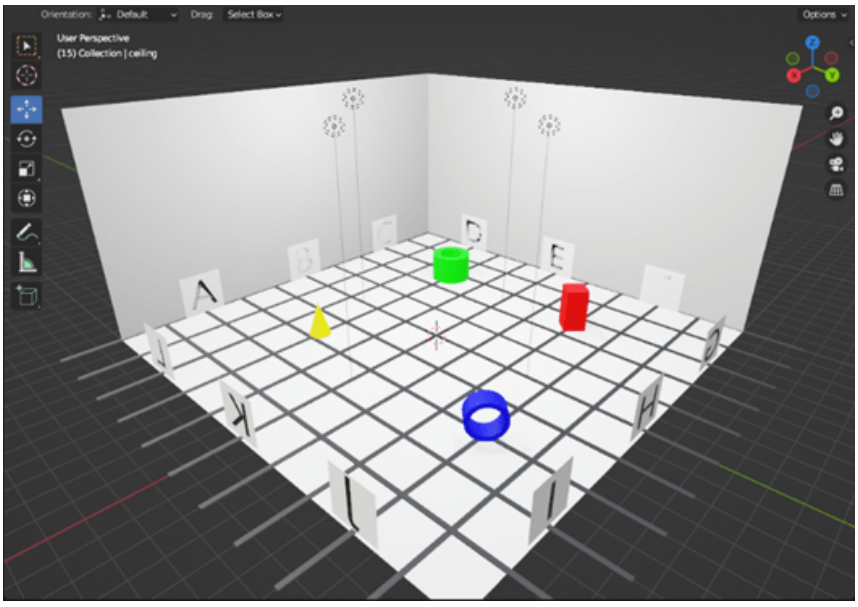


Figure 5.17: Representation of *Scene00_grid_signs* environment

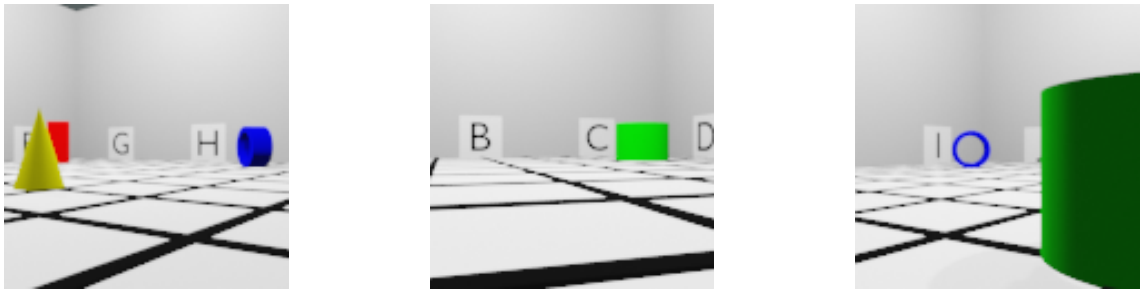


Figure 5.18: Examples of images rendered in *Scene00_grid_signs*

Contiuous Batch Learning - Average Distance 10000 epochs 2000 batches		
Scene	**XY3AF**	**XY3AF_stereo**
Scene00_lines	20.35	27.51
Scene00_grid	22.08	26.26
Scene00_grid_signs	13.63	19.30
Scene00_colored_grid_signs	5.54	9.08

Table 5.16: Results from adding objects to *Scene00* experiments

be, helps the model achieve better results in localization. By placing different types of markers, such as the signs, in ambiguous zones or zones without any distinguishable features, improved localization, meaning even if it doesn't improve as much as coloring the walls, the letter signs were helpful in improving the performance.

While not surpassing their performance in both *Scene00_colored* and *Scene00_stripe1*, the performance of both models in *Scene00_colored_grid_signs* still achieves localization with a reasonably lower distance error between ground truth and predicted values. In this case, the added

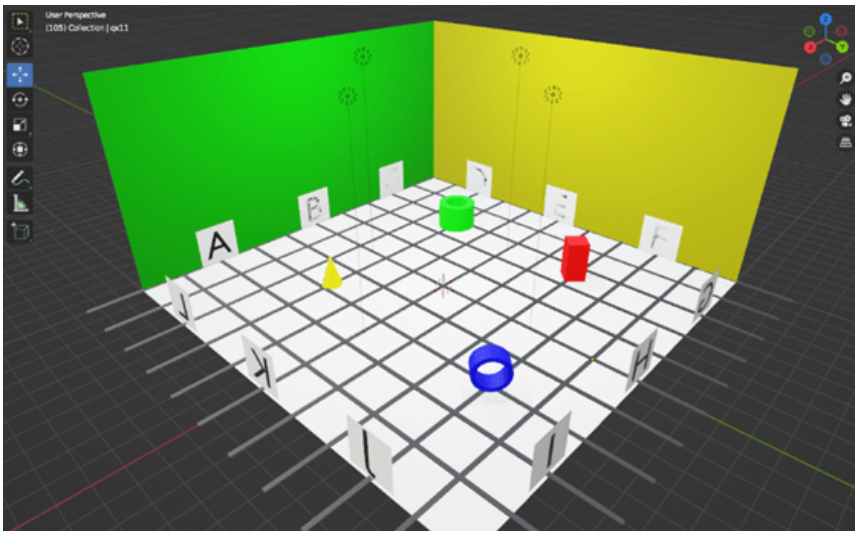


Figure 5.19: Representation of *Scene00_colored_grid_signs* environment

Figure 5.20: Examples of images rendered in *Scene00_grid_signs*

elements may not have contributed for localization as was expected, possibly even being detrimental to the performance. However, it may be the case that with the added elements and visual information, it warrants a longer training process than 10000 epochs.

5.7 Scene01 experiments

This part is dedicated to experiments conducted in the *Scene01* environment, which is a larger and more complex environment that seeks to simulate a small, simple warehouse while still being constructed from simple 3D meshes with not a lot of detail. This scene was described in more detail in section 3.3

With the added detail, the hope is that the latest iteration of the model, *XY3AF_stereo*, performs better compared to the previous scene. These experiments will evaluate the hypothesis that this model may thrive in more detailed and complex environments.

The conditions of the experiments, as well as the evaluation metric, are the same as in the previous experiments, allowing for a direct comparison in terms of performance. Since the size of *Scene01* is twice the size of *Scene00*, the best way to compare the accuracy is to determine what percentage of the scene's size the distance calculated is. The average distance error resulting from the experiments is showcased in table 5.17, and example images from this scene are displayed in 5.21.

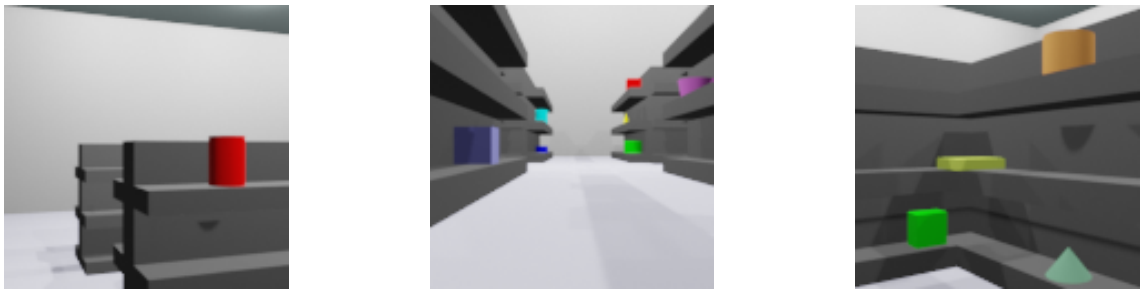


Figure 5.21: Examples of images rendered in *Scene01*

Continuous Batch Learning 10000 epochs 2000 batches	
Model	**Average Distance**
XY3AF	181.00
XY3AF_stereo	193.00

Table 5.17: Results from *Scene01* experiments.

The results were very disappointing, which wasn't expected from a scene with more elements. The added details should provide more useful information to the model, but this wasn't the case. A possible explanation for the lack of success is that the same lack of detail problem near the walls was the problem to be solved by the *stereo* version of the model.

To compare the performance with *Scene00*, the percentage error compared with the *Scene01* length (700) is 25.86% for *XY3AF* and 27.57% for *XY3AF_stereo*. These are far worse than the

ones obtained in section 5.3 for *Scene00* (17.18% for *XY3AF* and 19.90% for *XY3AF_stereo*), meaning the added complexities of this environment may have hindered the performance of the models.

Experiments were conducted in 5.6 that studied how to improve the environment to make it more suitable for localization. A simple modification that doesn't involve adding or changing the locations of objects is adding different colors to the walls, which helps distinguish them and aids in localization. With that being the case, to further demonstrate this point in a different environment, experiments are conducted that apply this change. The results of the experiments are presented in Table 5.18, and examples of the images rendered in this scene are displayed in 5.22.

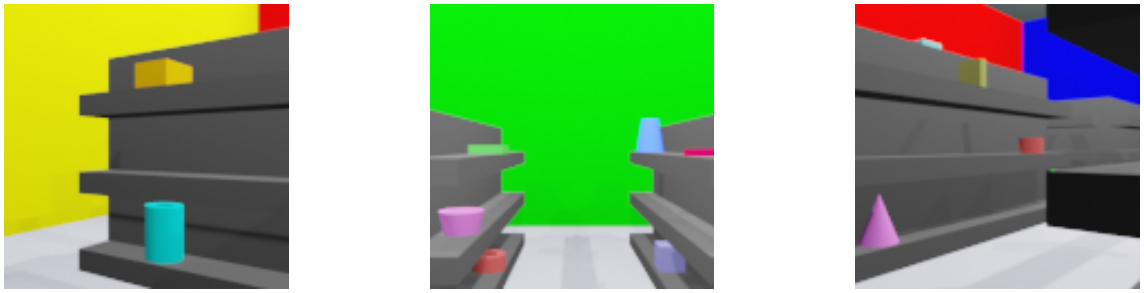


Figure 5.22: Examples of images rendered in *Scene01_colored*

Continuous Batch Learning 10000 epochs 2000 batches	
Model	**Average Distance**
XY3AF	30.95
XY3AF_stereo	46.07

Table 5.18: Results from *Scene01_colored* experiments.

The percentage of average error compared with *Scene01*'s length is 4.42% for *XY3AF* and 6.58% for *XY3AF_stereo*. These results can then be compared to those obtained in *Scene00_colored*, where the percentages of 2.66% for *XY3AF* and 3.96% for *XY3AF_stereo* showcase a better performance.

The colored walls greatly improved both models' performance, as was the case when the same change was applied to *Scene00*. The *XY3AF* model continues to provide better accuracy compared to its stereo counterpart, maintaining consistency with previous experiments. The continuous shortcomings of *XY3AF_stereo* indicate that the solution isn't superior to the original version, as was hoped.

Further experiments can be conducted to understand these shortcomings, and if there are certain scenarios where stereo surpasses *XY3AF*. Some possible future work that explores this issue is described in chapter 6.

5.8 Continuing Best Experiments

Some of the experiments conducted in this chapter have showcased great performances but did not reach their full potential, given the number of epochs the training ran for, which is a common limitation for every batch experiment. One of these models will continue to undergo training to determine if longer sessions can yield better results, and the outcome will be presented and discussed in this section.

For both *XY3AF* and *XY3AF_stereo* models, their best performance was achieved in *Scene00_stripe1*, where a colored stripe was added to each wall. Considering that *XY3AF* was always superior to the stereo version, the training will resume only with *XY3AF* in *Scene_stripe1* for an extra 10000 epochs. The result is displayed in the table 5.19. A heatmap that represents an overview map of the environment was constructed to visually observe the model's performance, and is presented in image 5.23.

Continuous Batch Learning 20000 epochs 2000 batches	
Model	**Average Distance**
XY3AF	2.83

Table 5.19: Results from the further training experiments of *XY3AF_stereo* in *Scene00_stripe1*, with a total of 20000 training epochs.

The performance of the model improved by a slight margin with the longer training process. From the heatmap, the improvement is noticeable when compared to the ones in section 5.2, but the error over the map isn't perceptible due to the low error compared to the scale used before. Another heatmap was then formed, now scaled to 28, which is the highest distance error. It is presented in image 5.24, and better represents the fluctuation of error over the environment.

The error increases by a significant margin when near the object present in the top right quadrant of the environment, which corresponds to the blue sideways tube. A visible but less significant error can also be seen near two other objects, the green tube and the yellow cone. The higher error that was visible near the walls in the heatmap in image 5.3 is no longer present, being more similar to the rest of the environment.

This improvement from the longer training session may also be the case for the stereo variation, but it wasn't conducted in this thesis because of the time needed to run it. In the next chapter 6, a paragraph is dedicated to further and longer experiments that aim to explore the full potential of *XY3AF_stereo*.

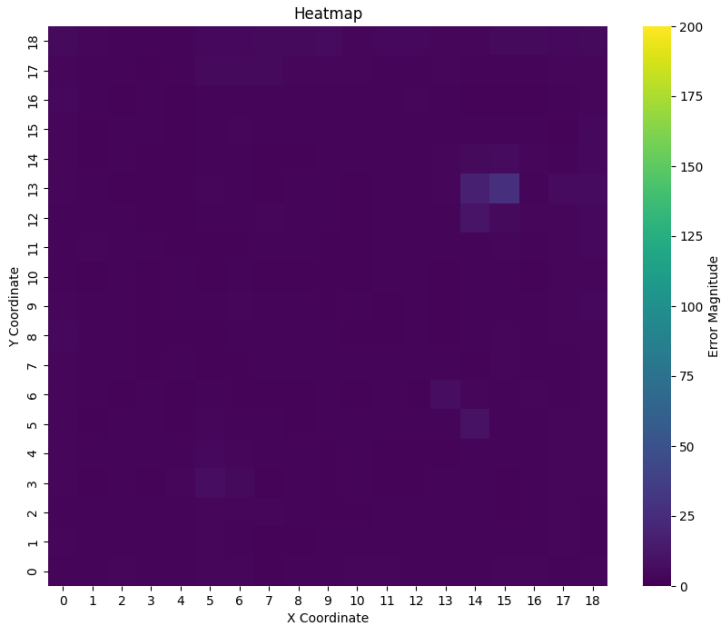


Figure 5.23: Average distance error heatmap of the best-performed *XY3AF* model, which can be interpreted as an above view of *Scene00_stripe1*

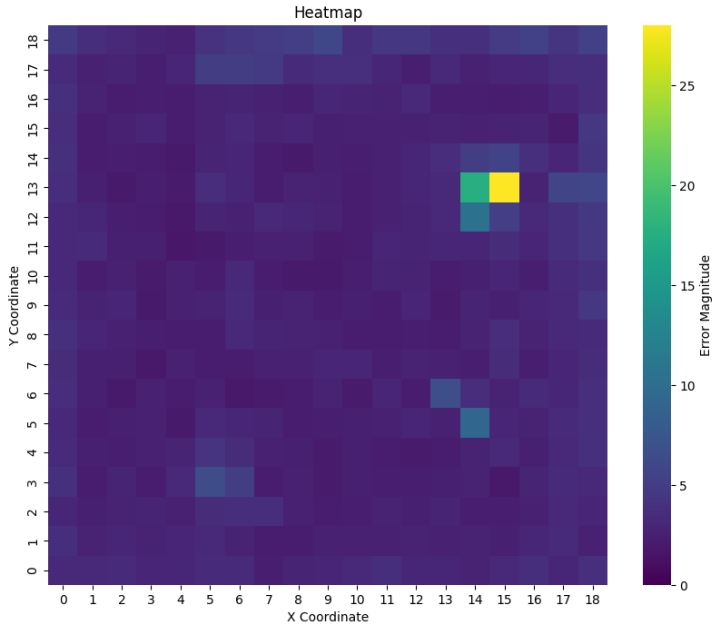


Figure 5.24: Same heatmap of the *Scene00_stripe1* model as in image 5.23, but scaled to the highest error, which is 28

Chapter 6

Future Work

This chapter outlines potential future explorations of the work done so far, including steps to achieve viable localization in more complex and detailed environments, as well as other possible iterations of the model that may yield better results.

The stereo model did not improve over the single-image version, so this approach may not be a correct assessment of the problem. However, this may be limited by two points: 1) the models need to train much longer compared to the single-image version because of the bigger amount of data that needs to be processed and understood; 2) the pair of images does not provide enough information to surpass the added dimension of complexity and obtain better results.

A possible approach that further explores this added dimension of images capturing more of the scene at once is to apply a four-image view, with a 90°rotation between images. While this may provide more information at once, the same problems associated with the stereo model proposed in this thesis will probably still be present.

Changes to the input might result in better localization performance, such as changing the FOV and especially the resolution. Increasing these aspects may provide a clearer view of the environment and capture more information overall. Also, generating a depth image dataset along with the RGB image datasets may provide more 3D information about the scene and allow a comprehensive understanding of the objects' positions. Altering the images, such as rotating the image or applying color filters to simulate different lighting, may also improve performance by avoiding overfitting.

In realistic applications, a robot would be able to input previously observed images, creating a sense of movement between them. Using a set of images that illustrate movement adds information that may be useful in the model being developed by avoiding the localization errors in ambiguous areas, such as the corners of the environment. The combination of images may be more easily localized compared to a static image.

One way to advance indoor localization is by achieving accurate positioning within a virtual, lifelike environment. The tools provided by *Blender* and *Python* coding developed in this project allow for the creation of such a realistic scene, which can create datasets that enable the training of localization models that could lead to practical applications in real-world scenarios. A warehouse environment is a great example to showcase this hypothesis, since it's a structured indoor

environment that can benefit significantly from robots equipped with self-localization and object detection capabilities, improving the efficiency of object retrieval. Importing STL files that translate into complex objects is a feature that was implemented, but not used in this project, which helps the objects depicted in the environment attain realism, and so better represent a realistic warehouse.

Exploring different machine learning architectures like ResNet, which is commonly used in image-based tasks, when fitted to use the stereo-image input, may also result in different performances compared to the VGG architecture employed in this project. While not being a focus of this project, since the creation of datasets from scratch was an important aspect of this thesis, employing pretrained models for performance comparison could offer valuable insights into the problem.

Further experiments on environmental changes would also provide a more complete insight into the limitations of the model developed and how to enhance it with minimal changes to the scene. One that was not conducted due to time constraints and that required a new reformulation of the model was one where visual markers would be placed on the ceiling, and an upward image capturing the ceiling would be rendered at every rendering point, serving as additional input to the model.

A feature that was developed but wasn't applied to the experiments is that additional data can be added to the training dataset while the training is still underway, and the model will use that data as input as well. This aspect allows us to evaluate the performance midway through the learning procedure, understand its shortcomings, and adapt the dataset to conform to those shortcomings, for example, adding more images of specific areas or even adding or changing elements of the scene.

Chapter 7

Conclusion

This thesis offers an extensive study of image-based machine learning architectures *Convolutional Neural Networks* and the specialized *VGG* architecture, applying it to localization methods in artificial environments. Different learning models were developed to achieve localization, employing single and two-image (stereo) input, and different output representations of the model.

The experiments regarding the stereo variation of the model, unfortunately, did not result in the expected outcome, where it would improve upon the single image version. However, the single image variation of the *XY3AF* model resulted in very positive results, reaching an 2,83 average distance error between the real and predicted values in the *Scene00_stripe1* environment.

Even with the shortcomings of the model developed, this whole project still was useful in understanding the inner workings of this kind of solution, while providing and studying tools, like *Blender*'s *Python* API, that enable and facilitate the creation of quality image datasets. The experiments conducted allowed for an exploration of environmental improvements that showcased the influence of certain elements and demonstrated their beneficial impact on the model's performance.

The impact of the training datasets on the model's performance was also showcased in this work. Adding elements to the scene, such as the lines on the floor, overall helped in the model's performance, but by far, the presence of color was much more beneficial in differentiating locations in the environment. Applying these changes to featureless zones that are captured in a majority of the input images, which in the case of this project were the gray walls, greatly improves performance. Even with a slight distinctive feature, such as the very light color that was added to the gray walls in one of the experiments, greatly improves localization.

These additional features were needed for the models to achieve any acceptable performance, meaning that for the main basic scene studied in this project (*Scene00*), the models developed couldn't locate themselves accurately. Although it is a more expensive process to apply these kinds of change to real-life scenarios, they are beneficial to localization models and may be worth the investment.

This work also proposes future work that further explores the tools and architectures exposed, as well as different approaches to achieve localization in artificial environments, and possible real-world applications.

Bibliography

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

[2] Abitya Bagaskara and Muhammad Suryanegara. Evaluation of vgg-16 and vgg-19 deep learning architecture for classifying dementia people. In *2021 4th International Conference of Computer and Informatics Engineering (IC2IE)*, pages 1–4, 2021.

[3] Dor Bank, Noam Koenigstein, and Raja Giryes. *Autoencoders*, pages 353–374. Springer International Publishing, Cham, 2023.

[4] Abdelhak Bougouffa, Emmanuel Seignez, Samir Bouaziz, and Florian Gardes. Smarttrolley: An experimental mobile platform for indoor localization in warehouses. In *2020 3rd International Conference on Robotics, Control and Automation Engineering (RCAE)*, pages 108–115, 2020.

[5] Marcus A. Brubaker, Andreas Geiger, and Raquel Urtasun. Map-based probabilistic visual self-localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(4):652–665, 2016.

[6] Lukas Budach, Moritz Feuerpfeil, Nina Ihde, Andrea Nathansen, Nele Noack, Hendrik Patzlaff, Hazar Harmouch, and Felix Naumann. The effects of data quality on ml-model performance, 07 2022.

[7] 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 *2017 International Conference on 3D Vision (3DV)*, pages 667–676, 2017.

[8] Changhao Chen, Bing Wang, Chris Xiaoxuan Lu, Niki Trigoni, and Andrew Markham. A survey on deep learning for localization and mapping: Towards the age of spatial machine intelligence. *CoRR*, abs/2006.12567, 2020.

[9] Changhao Chen, Bing Wang, Chris Xiaoxuan Lu, Niki Trigoni, and Andrew Markham. Deep learning for visual localization and mapping: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(12):17000–17020, 2024.

[10] Davide Chicco. *Siamese Neural Networks: An Overview*, pages 73–94. Springer US, New York, NY, 2021.

[11] Jiyong Chung and Keemin Sohn. Image-based learning to measure traffic density using a deep convolutional neural network. *IEEE Transactions on Intelligent Transportation Systems*, 19(5):1670–1675, 2018.

[12] Vasco Cruz. An api for building artificial worlds for machine learning using blender. `https://drive.google.com/file/d/1AQ_2hWuL_jNr89gnVDxefZD6i2CIb2hG/view`, 2022. [Thesis].

[13] Youdi Gong, Guangzhen Liu, Yunzhi Xue, Rui Li, and Lingzhong Meng. A survey on dataset quality in machine learning. *Information and Software Technology*, 162:107268, 2023.

[14] Stephen Hausler, Adam Jacobson, and Michael Milford. Multi-process fusion: Visual place recognition using multiple image processing methods. *IEEE Robotics and Automation Letters*, 4(2):1924–1931, 2019.

[15] Gijeong Jang, Sungho Kim, Wangheon Lee, and Inso Kweon. Robust self-localization of mobile robots using artificial and natural landmarks. In *Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation. Computational Intelligence in Robotics and Automation for the New Millennium (Cat. No.03EX694)*, volume 1, pages 412–417 vol.1, 2003.

[16] Vimuthki Jayawardene, Shazia Wasim Sadiq, and Marta Indulska. An analysis of data quality dimensions. 2015.

[17] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of Deep Learning-Based Object Detection. *IEEE Access*, 7:128837–128868, 1 2019.

[18] Longlong Jing and Yingli Tian. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):4037–4058, 2021.

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

[20] Shuang Li, Baoguo Yu, Yi Jin, Lu Huang, Heng Zhang, and Xiaohu Liang. Image-based indoor localization using smartphone camera. *Wireless Communications and Mobile Computing*, 2021:1–9, 07 2021.

[21] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, 2022.

[22] Liu Liu and Hongdong Li. Lending orientation to neural networks for cross-view geolocalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[23] Yang Long, Yiping Gong, Zhifeng Xiao, and Qing Liu. Accurate object localization in remote sensing images based on convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 55(5):2486–2498, 2017.

[24] Wei Luo, Jun Li, Jian Yang, Wei Xu, and Jian Zhang. Convolutional sparse autoencoders for image classification. *IEEE Transactions on Neural Networks and Learning Systems*, 29(7):3289–3294, 2018.

[25] Iaroslav Melekhov, Juha Ylioinas, Juho Kannala, and Esa Rahtu. Image-based localization using hourglass networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017.

[26] Curtis Northcutt, Lu Jiang, and Isaac Chuang. Confident learning: Estimating uncertainty in dataset labels. *J. Artif. Int. Res.*, 70:1373–1411, 2021.

[27] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.

[28] S. Picard, C. Chapdelaine, C. Cappi, L. Gardes, E. Jenn, B. Lefevre, and T. Soumarmon. Ensuring dataset quality for machine learning certification. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 275–282, 2020.

[29] Noha Radwan, Abhinav Valada, and Wolfram Burgard. Vlocnet++: Deep multitask learning for semantic visual localization and odometry. *IEEE Robotics and Automation Letters*, 3(4):4407–4414, 2018.

[30] R. M. M. R. Rathnayake, Madduma Wellalage Pasan Maduranga, Valmik Tilwari, and Maheshi B. Dissanayake. RSSI and Machine Learning-Based Indoor Localization Systems for smart cities. *Eng—Advances in Engineering*, 4(2):1468–1494, 5 2023.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[32] Claudio Filipi Gonçalves Dos Santos and João Paulo Papa. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. *ACM Comput. Surv.*, 54(10s), 2022.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[34] Emiliano Spera, Antonino Furnari, Sebastiano Battiato, and Giovanni Maria Farinella. EgoCart: a benchmark dataset for Large-Scale indoor Image-Based localization in retail stores. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(4):1253–1267, 9 2019.

[35] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The replica dataset: A digital replica of indoor spaces, 2019.

[36] Paolo Tripicchio, Salvatore D'Avella, and Matteo Unetti. Efficient localization in warehouse logistics: a comparison of lms approaches for 3d multilateration of passive uhf rfid tags. *The International Journal of Advanced Manufacturing Technology*, 120(7):4977–4988, Jun 2022.

[37] Rahul Rama Varior, Mrinal Haloi, and Gang Wang. Gated siamese convolutional neural network architecture for human re-identification. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 791–808, Cham, 2016. Springer International Publishing.

[38] S. Vignesh, M. Savithadevi, M. Sridevi, and Rajeswari Sridhar. A novel facial emotion recognition model using segmentation VGG-19 architecture. *International Journal of Information Technology*, 15(4):1777–1787, 3 2023.

[39] Yihong Wu, Fulin Tang, and Heping Li. Image-based camera localization: an overview. *Visual Computing for Industry, Biomedicine, and Art*, 1(1):8, Sep 2018.

[40] Ping Zheng, Danyang Qin, Jianan Bai, and Lin Ma. Image-based indoor localization using smartphone cameras in textureless environment. In *2023 6th International Conference on Information and Computer Technologies (ICICT)*, pages 226–231, 2023.

[41] Xu Zhong, Yu Zhou, and Hanyu Liu. Design and recognition of artificial landmarks for reliable indoor self-localization of mobile robots. *International Journal of Advanced Robotic Systems*, 14(1):1729881417693489, 2017.