DEEP LEARNING IN GRAPH DOMAINS FOR SENSORISED ENVIRONMENTS

DANIEL RODRIGUEZ CRIADO

Doctor of Philosophy

ASTON UNIVERSITY

August 2023

Abstract

As our society moves swiftly towards an era where technology seamlessly integrates into our daily lives, our homes and cities are becoming increasingly sensorised. This change is fueled by advancements in artificial intelligence that facilitate harnessing the potential of *smart* environments. The main focus of this thesis is to investigate how Graph Neural Networks (GNNs) can be effectively applied to these environments, with a focus on those where humans and robots share the space. In these scenarios, integrating and exploiting data from multiple sources and analysing interactions between individuals, objects, sensors and robots is paramount. As the literature shows, GNNs have advantageous properties to process this kind of data when compared to more established deep learning approaches.

This thesis presents a range of methods and applications in sensorised environments that leverage GNNs' properties. The main contributions span applications in three main fields: human-aware robot navigation, human pose estimation, and the generation of traffic images. For human-aware navigation, this thesis proposes a model capable of estimating the level of discomfort caused by a robot's presence among people and objects, considering not only the entities themselves but also the interactions happening. This model is later improved to yield discomfort maps that can be used as cost maps for motion planning. In the domain of human pose estimation, two different solutions are presented: a model capable of estimating the position and orientation of the people in the environment, and a multi-camera and multi-person 3D human full pose estimator. This last model, which does not require a labelled dataset for training, can be used for tracking people and feed their poses into the aforementioned cost map generator, as seen in the experimentation of this thesis. These works exhibit superior results in terms of precision, accuracy, and time efficiency when compared to similar state-of-the-art works.

Finally, in the field of image generation, the thesis explores an application within the context of *smart* cities: generating realistic traffic images conditioned with graphs. This work leverages the strengths of GNNs when working with semantic data. The model can generate realistic images based on the properties of the items expected in them –namely their position, size and colour– and global properties such as the time of day.

GNNs can be time-inefficient due to the added complexity of dealing with heterogeneously structured data. Consequently, the success of the applications presented in this thesis is the result of the effective integration of this networks, often in conjunction with other well-known approaches. One notable example is the fusion of convolutional networks with GNNs, which in this thesis leads to more efficient image generation when compared to pure GNN architectures. These methods constitute the central contribution of this thesis, as they allow GNNs to fully exploit their potential while mitigating inefficiencies.

Keywords: Graph Neural Networks, Sensorised Environments, Human-Aware Navigation, Multi-Camera Pose Estimation, Robotics, Image Generation.

To my family,

for you constant love and support.

Acknowledgements

The culmination of this thesis marks the end of an extraordinary four-year chapter of my life, punctuated by exceptional experiences and formidable challenges. The most daunting perhaps was maintaining focus and productivity during the COVID-19 pandemic. I was fortunate, however, to have the support of numerous friends whose guidance and assistance rendered this PhD journey not only feasible, but also enriching.

Foremost, I express my great gratitude to Dr Luis J. Manso, my supervisor and enabler of this academic adventure. Luis, with his vast expertise spanning robotics, computer science, AI and much more, served as an invaluable mentor from whom I learned a lot. His unwavering work ethic, exceptional problem-solving abilities, and supportive nature have been an inspiration throughout my PhD.

To my associate supervisors, Dr George Vogiatzis and Dr Maria Chli, I extend my profound appreciation. George's ability to contribute insightful ideas consistently resulted in enhancements to the projects constituting my thesis. Maria, your generosity in allowing me to take part in the Traffic3D project was a rewarding experience that broadened my thesis horizons significantly; I am immensely grateful for this opportunity.

This PhD journey would have been considerably less fruitful without the priceless contributions of Dr Pilar Bachiller. Pilar, your remarkable ability for bug detection in the code, your intuitive problem-solving approach, and your tireless dedication to work have left an indelible impression on me.

My gratitude also extends to my friends in the Computer Science department - Renato Arantes, Thomas Carr, and Juan Marcelo Parra. Your friendship since the start of this thesis and the memorable times we shared are irreplaceable. Additionally, Nelson Castro deserves special mention; although hailing from a different department, our intellectually invigorating discussions have provided many worthwhile insights for my work.

A heartfelt thanks to all my friends in Birmingham who have augmented this transformative journey. Your presence has undoubtedly enriched my experience in this city.

Lastly, I offer my deepest thanks to my family - my parents, Sebastian and Mercedes, and my brother, Alberto. Your unwavering belief in me and the strength you provided during my most challenging moments have been truly uplifting.

Sara, your firm support has been an integral part of this journey, particularly during the thesis' last and most demanding stages. Your continuous care and encouragement were a pillar of strength that made this arduous journey considerably smoother. For this, I extend my heartfelt thanks to you.

List of Publications

Publications Arising from this Thesis:

2020:

[145] **Rodriguez-Criado, Daniel**, P. Bachiller, P. Bustos, G. Vogiatzis, and L. J. Manso. Multi-camera torso pose estimation using graph neural networks. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 827–832. IEEE, 2020.

[11] R. Baghel, A. Kapoor, P. Bachiller, R. R. Jorvekar, **Rodriguez-Criado, Daniel**, and L. J. Manso. A toolkit to generate social navigation datasets. In *Workshop of Physical Agents*, pages 180–193. Springer, 2020. **(Best paper award.)**

> My main contributions to this paper were drafting the Introduction and Related Work sections, and developing the simulation controller.

2021:

[146] **Rodriguez-Criado, Daniel**, P. Bachiller, and L. J. Manso. Generation of human-aware navigation maps using graph neural networks. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 19–32. Springer, Cham, 2021. **(Best student paper award.)**

[10] P. Bachiller, **Rodriguez-Criado, Daniel**, R. R. Jorvekar, P. Bustos, D. R. Faria, and L. J. Manso. A graph neural network to model disruption in human-aware robot navigation. *Multimedia Tools and Applications*, pages 1–19, 2021.

> My responsibilities for this paper were parallel to those of the first author. I concentrated on model optimization and generating various versions of the graphs. Additionally, I contributed to the drafting and editing of the entire paper.

2023:

[147] **Rodriguez-Criado, Daniel**, M. Chli, G. Vogiatzis, and L. J. Manso. Synthesizing traffic datasets using graph neural networks. In *International Conference on Intelligent Transportation Systems*. IEEE, 2023.

Publications under review:

Part of the work presented in Chapter 6 is currently under submission to the *International Journal of Computer Vision*, titled "Multi-person 3D Pose Estimation from Unlabelled Data". Authors: **Daniel Rodriguez-Criado**, Pilar Bachiller, George Vogiatzis, and Luis J. Manso.

Research from Chapter 5 is planned for submission to the *Journal of Ambient Intelligence and Humanized Computing* under the title "*Generation of Dynamic Human-aware Navigation Maps using Graph Neural Networks*". Authors: **Daniel Rodriguez-Criado**, Pilar Bachiller, and Luis J. Manso.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In our rapidly evolving world, *smart* environments are becoming ubiquitous. These environments are physical domains or ecosystems that dynamically interact with their inhabitants, these environments collate and analyse data, autonomously adapting to alterations and occurrences. They leverage technologies such as the Internet of Things (IoT) and edge computing [112], supported by advancements in telecommunications (fifth-generation 5G networks) and sensor technologies, as well as Artificial Intelligence (AI) and data analytics to optimise energy efficiency, enhance comfort, improve safety, and deliver an upgraded user experience. *Smart* environments can be found in various contexts, including smart homes, smart cities, smart buildings, and smart transportation systems [6].

Inherently, *smart* environments must be sensorised. For the purposes of this thesis, a √ ***sensorised environment*** can be defined as a setting wherein a multitude of sensors and often actuators collaborate to provide an agent with comprehensive information about its surroundings. For instance, a room equipped with multiple cameras capturing images from diverse angles and transmitting them to a robot via wireless connection can be regarded as a sensorised environment. It can also refer to a city equipped with CCTV cameras, air quality sensors, illumination sensors, and other monitoring devices. The gathered information must be fused and processed to extract key features that will be used for further actions. In this task, it would be interesting to use algorithms capable of learning and extracting the most useful features from the data. This thesis explores the design and application of learning models in these sensorised environments.

Regarding indoor sensorised environments, they are expected to incorporate mobile and assistive robots, which can assist with a multitude of crucial functions, such as companionship [64], aiding elderly individuals with chores, or supporting people with disabilities [33]. Navigation constitutes one of the most fundamental skills required for robots to be functional, as they need to manoeuvre within their environment to carry out basic tasks. In order to be seamlessly integrated into our daily lives, robots should adhere to social conventions, thereby mitigating any potential disruption to humans.

Over the past few decades, there has been significant research in the areas of proxemics and interpersonal distance acceptance for humans, leading to the emergence of the concept √ of ***Human-Aware Navigation (HAN)***. As outlined by Kruse et al. [99], three primary requirements for socially acceptable robot navigation are comfort, naturalness, and sociability. These concepts are further explored in Chapter 4. Historically, literature on robot navigation primarily focused on collision avoidance with objects, humans, and walls. However, HAN seeks to go a step further by taking into consideration the social interactions (human-human and human-robot) as well as respecting the interpersonal space.

Pioneering work taking into account human awareness in robot navigation used hard-coded features to create the path followed by the robot in the presence of humans [135], [91]. The main drawback of this technique is that social behaviour or human behaviour is stochastic. Creating a hand-crafted program for a robot to predict human actions is challenging, as there is, arguably, too many variables to handle manually [178]. Instead, learning through demonstration proves more effective. Many contemporary approaches, such as [37], [30], [137], or [139], employ machine learning models to address this issue, allowing an agent to learn from experiences facilitated by a human. Consequently, Chapters 4 and 5 in this thesis delineate the development of learning algorithms tailored for HAN. Specifically, these chapters delve into the creation of models for estimating the discomfort induced by a robot in a given space and generating corresponding cost maps based on these discomfort scores for robot navigation.

In the realm of robot navigation, environmental awareness is crucial for the development of robots that can learn from their surroundings and adapt their behaviour in response to changes. Part of the research conducted in this thesis aimed to develop algorithms and applications for robots operating in sensorised environments where they must coexist with humans. Thus, RGB cameras, which are relatively inexpensive sensors, can be employed to detect humans in a sensorised environment. Numerous studies have investigated the detection of 2D human poses using RGB camera sensors [26, 192]. However, it is required to estimate the position of people within a 3D space, rather than merely in the image plane of the cameras, in order to provide meaningful information to the robot. ***3D Human Pose Estimation (HPE)*** can prove to be a complex task, particularly when using a multi-camera setup in which information from various cameras must be fused to determine final 3D poses (additional information about this problem can be found in Chapter 6). Traditional approaches address the issue by employing geometric and appearance cues, as well as epipolar geometry [46, 18]. Nonetheless, these approaches, which rely on analytic algorithms, often struggle to handle occluded body parts. For instance, a 3D position cannot be obtained through triangulation if it is not detected by at least two cameras. In such cases, a machine learning algorithm could potentially learn to utilise the available information to "hallucinate" or infer the missing body parts when detections from multiple cameras are absent. This approach would allow the algorithm to fill in the gaps and estimate 3D poses more accurately. Two learning-based approaches for human orientation estimation and 3D HPE are presented in Chapter 6.

This thesis also explores outdoor sensorised environments, such as those found in *smart* cities. Traffic congestion, especially prevalent in major global cities, is a persistent issue exacerbated by the growing number of vehicles. In this setting, effective traffic management is critical to reduce travel delays, road accidents, and environmental pollution. ***Intelligent Transportation Systems (ITS)*** integrate sensing and communication technologies with automatic control techniques, with the aim of augmenting the safety and efficiency of transportation infrastructure.

Traffic junctions are pivotal points in traffic management as they function as shared physical spaces traversed by numerous vehicles. Effective traffic light control at these intersections can lead to improved traffic flow. Traditional traffic lights, often based on handcrafted algorithms, prove inefficient as they struggle to adjust dynamically to variable traffic patterns. Advancements in machine learning offer a solution through the use of algorithms that deduce optimal policies from raw sensory data such as traffic images. However, this solution requires the collection of a substantial amount of data for model training.

Chapter 7 addresses this issue by offering a tool for generating artificial traffic images using machine learning. This tool facilitates the generation of large datasets for training not only traffic control systems, but also other traffic applications based on raw images, such as traffic surveillance and traffic prediction among others.

In the previously outlined projects, purely analytic solutions demonstrate limitations, particularly with increased problem complexity or missing information. Machine learning and deep learning algorithms have been proposed as more adept at addressing these limitations. Before proceeding, it is essential to briefly define some fundamental terms, which will enable the reader to better comprehend the scope of the work and the positioning of the deep learning paradigm within the broader context of AI.

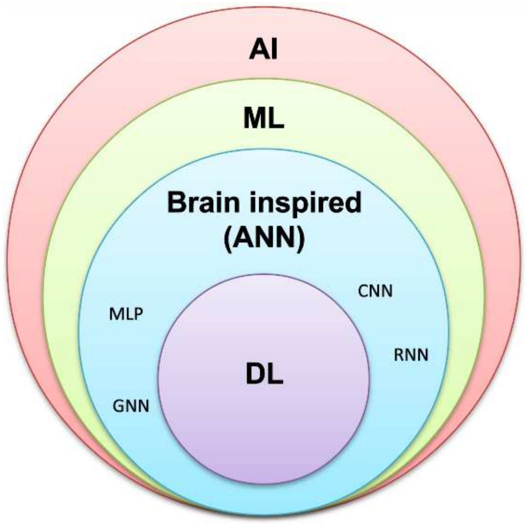


Figure 1.1: Diagram outlining the scope of deep learning within the broad frame of artificial intelligence.

Firstly, how is the term Artificial Intelligence (AI) understood nowadays? From the creation of the first computer, humans have regarded it as an intelligent agent, capable of performing tasks that were previously unattainable. Over time, these tasks have come to be known as computer programs. However, it soon became apparent that certain tasks, which humans can effortlessly execute, such as visual recognition, speech understanding, and reasoning, are nearly impossible for a programmer to code into a machine. This realisation led to a complete redefinition of artificial intelligence [172]. It is now better understood as the “simulation of human intelligence processes by machines, particularly computers” [155]. This definition is one of many, as the term AI encompasses a vast umbrella of fields and purposes. In this line, contemporary AI research aims to emulate, and in some instances, surpass human capabilities in various tasks.

Machine Learning (ML) is a subset of AI that enables machines to learn from observations and data without explicit programming. Unlike conventional AI models, ML algorithms can learn autonomously, training themselves using samples of inputs and expected outputs. √ ***Deep Learning (DL)*** is a subset of ML that utilises deeper networks with a larger number of parameters. As a result, more data is needed to train these models, but they extract higher-level features from the input, yielding improved results and generalisations. This allows for the creation of end-to-end applications directly from raw data, eliminating the need for handcrafted features for the inputs [59]. Consequently, the majority of models developed in this thesis fall within the scope of Deep Learning. Figure 1.1 illustrates a diagram that clarifies the positioning of Deep Learning within the framework of AI.

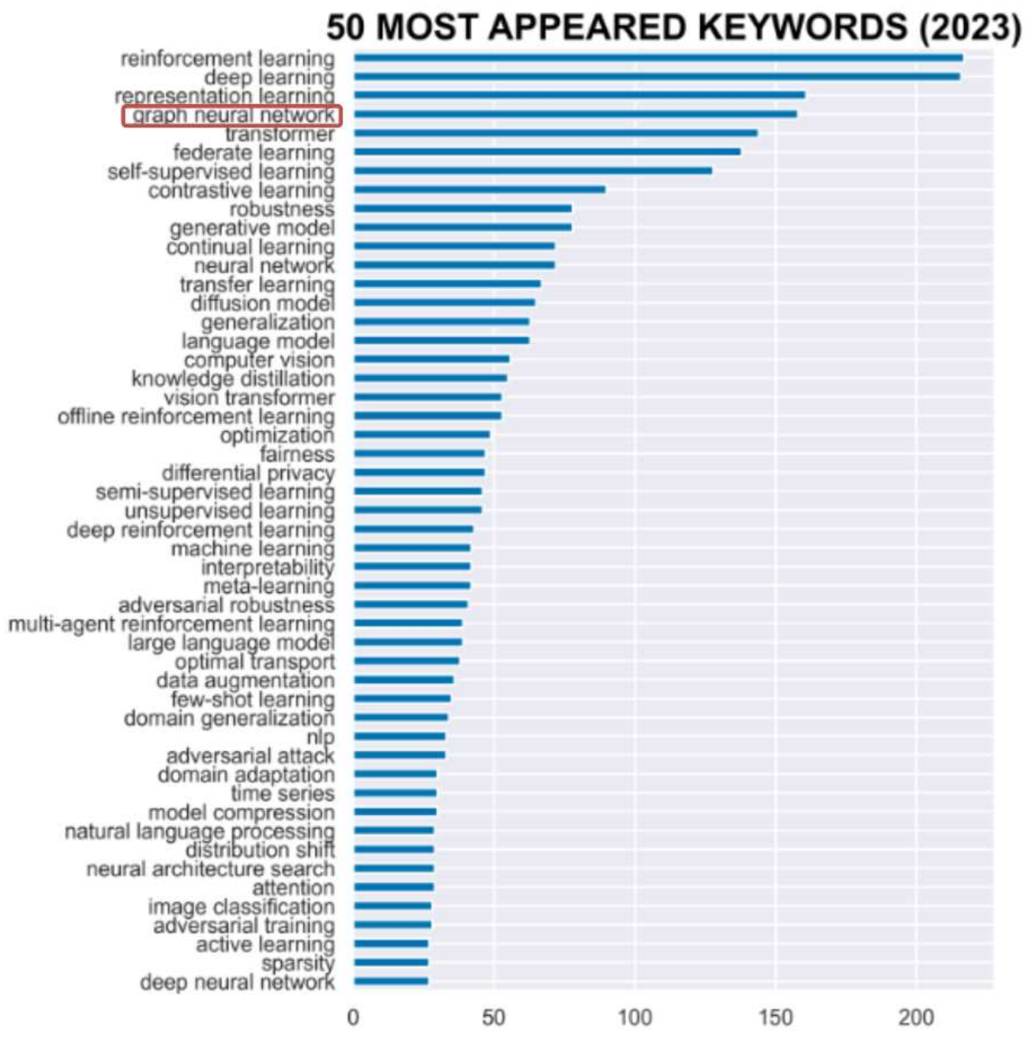


Figure 1.2: Keywords in ICLR articles 2023. Graph Neural Network is the 4th more mentioned keyword.[1].

[1]Source: `https://github.com/EdisonLeeeee/ICLR2023-OpenReviewData`

Within the diverse range of deep learning paradigms, the work presented in this thesis primarily focuses on the study of a relatively novel family of techniques known as Graph Neural Networks (GNNs) in conjunction with other state-of-the-art DL models (refer to Chapter 2 for more information about the fundamentals of DL). GNNs offer several advantages compared to traditional Euclidean structured methods, which will be discussed in Chapter 3. Although relatively new, GNNs demonstrate strong potential and have emerged as one of the hottest research topics in machine learning. As evidence of their popularity, Figure 1.2 depicts the most frequently occurring keywords in the ICLR 2023 conference, with Graph Neural Networks ranking among the most prominent.

To summarise, this thesis delves into enhancing deep learning applications leveraging GNNs across three key research domains in sensorised environments: human-aware navigation with robots, human pose estimation and image generation for data augmentation. The motivation for the choosing of these areas comes from their utility in our society and its numerous applications as discussed above. The bulk of the work concentrates on indoor sensorised environments for social and human-aware navigation with a robot and the fusion of camera sensors for detecting people's poses in the scene (the two first areas of reasearch). Chapter 7 explores the design of a model using an outdoor surveillance camera for image generation, employing a combination of GNNs and image processing DL models. Although this last project is not as closely related as the two previous areas of research, it utilizes GNNs combined with CNNs for efficient image generation, which is one of the key contributions of the previous projects. Furthermore, akin to the previous research areas, this last one operates within sensor-rich environments. The fundamental difference lies in the outdoor spatial context in this instance.

The thesis exclusively employs GNNs or their fusion with cutting-edge ANNs across all developed learning models to push the boundaries of the state-of-the-art within their respective application domains. This commitment is validated through meticulously designed experiments aimed at testing and comparing the performance of these innovative models against the prevailing benchmarks. Consequently, the thesis not only seeks to enhance specific applications in the aforementioned research domains but also aims to pioneer novel techniques for the more efficient utilization of GNNs. The ultimate objective lies in demonstrating the substantial advantages GNNs offer for certain applications within sensor-rich environments, provided they are effectively implemented. Sensorised environments in general, and specifically those presented in this thesis, are physical environments where higher level data and raw sensory inputs meet. Throughout the contributions of the thesis, evidence is provided supporting that GNNs have potential to exploit data at different levels of abstraction. In addition to dealing with heterogeneously abstract data, GNNs are order and size equivariant, which are beneficial for sensorised environments as, for instance, the number of cars or pedestrians vary. Further elaboration on the advantages of GNNs in sensor-rich environments can be found in Section 3.3 of Chapter 3. This advantages motivates their use and exploration in the applications of this thesis.

The research questions this thesis seeks to address are outlined in the following section 1.1. Throughout the research process, significant contributions have emerged, as detailed in section 1.2.

1.1 Research Questions

Having established the prevalent role of sensorised environments in contemporary society and that Graph Neural Networks could serve as powerful tools when applied to this domain, the experiments and applications developed throughout this thesis seek to support this statement. To rigorously examine the application of graph-based learning models in sensorised environments, this thesis sets out to address two primary research questions:

Q1. ***What are the benefits of Graph Neural Networks in comparison to classical approaches in this field?***

This global research question serves as the focal point of the thesis. To address it, various applications of GNNs in sensorised environments have been proposed, specifically focusing on perception, prediction, and planning in the robotics field, as well as the generation of realistic images from graphs. These particular applications have been selected due to their anticipated substantial impact on society, especially considering the escalating prevalence of sensorised environments. The intent is to delve into areas where GNNs can play a pivotal role, addressing the challenges and opportunities posed by these environments.

The experiments conducted for these applications demonstrate that GNNs capitalise on their inductive bias for graph-structured data and relational information, resulting in superior performance compared to state-of-the-art deep learning approaches. This superiority is observed, for example, in the generation of cost maps accounting for interactions among people, objects, and robots, (see Chapters 4 and 5) or in the fusion of multi-camera sensor data for estimating 3D human poses (see Chapter 6). Additionally, the integration of semantic data into graph features offers a significant advantage, as this information proves valuable in conditioning the generation of images, as demonstrated in Chapter 7.

Throughout this thesis, it will be demonstrated that the proposed GNN-based models effectively address some of the limitations of conventional deep learning approaches for specific applications.

Q2. ***How can Graph Neural Networks be efficiently applied to sensorised environments and robots, where working with structured and non-structured data is crucial? Can they be improved in any way for the specific applications of this thesis?***

One of the aims of this thesis is to study whether GNNs can outperform classical approaches in certain applications in sensorised environments. These networks are powerful in tasks such as fusing information from multiple sources, effectively exploiting data relationships, and capturing semantic data from graphs. However, GNNs can be computationally inefficient due to the added complexity of dealing with heterogeneously structured data. Consequently, the success of the applications presented in this thesis is the result of the effective integration of these networks, often in conjunction with other well-known approaches in DL. One notable example is the fusion of convolutional networks with GNNs, which in this thesis leads to more efficient image generation when compared to pure GNN architectures. These methods constitute the central contribution of this thesis, as they allow GNNs to fully exploit their potential while mitigating inefficiencies.

1.2 Contributions of this Thesis

This research critically investigates the integration of GNNs into applications within sensorised environments. It involves a series of extensive experiments aimed at exploring this integration, yielding valuable insights across multiple domains. Specifically, these experiments shed light on human-aware navigation with robots, the fusion of data from diverse sensors for 3D human pose estimation, and the utilization of semantic data to enhance realistic image generation.

The outcomes of these experiments provide a better understanding of Graph Neural Networks and also introduce improvements never seen in these models until the present time, which constitutes one of the most important scientific contributions of this thesis.

Furthermore, considering the escalating prevalence of sensorised environments in our society, the models and applications derived from this research possess the potential for substantial real-world impact in our daily lives.

Outlined below are the primary technological and scientific contributions of this thesis, offering a more detailed perspective:

- A dataset for human-human and human-robot interactions in a simulated environment, *SocNav2*, which incorporates 3D images, entity velocities, and trajectories, unlike its predecessor *SocNav1*. Additionally, a tool called *SONATA* has been developed to facilitate dataset creation. *SocNav2* was created using this tool, providing flexibility for designers. The dataset has been employed to train several models presented in this work. (Chapter 5). These contributions have notably led to the publications [10] and [11].

- A model, *SNGNNv2*, capable of predicting discomfort scores reflecting the extent to which a robot disturbs people in a room. These scores can be utilised to evaluate the performance of human-aware navigation algorithms, as the model has been trained using data extracted from real humans labelling these scenarios. (Chapter 5). This contribution has been documented in [10].

- Two distinct models, *SNGNN-2D* and *SNGNN-2Dv2*, developed to generate 2D cost maps from static and dynamic data, respectively. These maps represent the disruption areas caused by a robot to humans in a room and can be applied to human-aware navigation. Experiments with a real robot demonstrate the efficacy of these maps for navigation. (Chapters 4 and 5). *SNGNN-2D* has been documented in [146], while *SNGNN-2Dv2* is planned for forthcoming publication in the *Journal of Ambient Intelligence and Humanized Computing* under the title “Generation of Dynamic Human-aware Navigation Maps using Graph Neural Networks”.

- A novel modification to the input graphs of a GNN tailored for specific applications, facilitating seamless integration with Convolutional Neural Networks (CNNs) for image generation. These innovative graphs incorporate a lattice of nodes that can be effortlessly transformed into an image to be processed by the CNN. (Chapters 4, 5 and 7). The impact of this contribution is underscored by its inclusion in publications [146] and [147].

- A robust model capable of estimating human orientation and torso position in a plane, by efficaciously fusing data from multiple RGB camera sensors. The model boasts the ability to be trained with a hybrid dataset consisting of both simulated and real-world data, thereby enhancing its generalisability and practical utility. (Chapter 6). This contribution is published in [145].

- An advanced multi-camera, multi-person 3D full pose estimation model, which can be trained with unlabeled data using self-supervised learning techniques. Through empirical experiments conducted with a real robot, this model is demonstrated to be scenario-agnostic, effectively working with cameras mounted on a mobile robot and avoiding the need for scenario-specific training. (Chapter 6). This contribution, submitted to the *International Journal of Computer Vision*, is currently under review with the title“Multi-person 3D Pose Estimation from Unlabelled Data”.

- A unique combination of a GNN and an image generation model to produce realistic images of traffic crossroads. The GNN plays an essential role in conditioning the generated images with both metric and semantic data such as the coordinates of the vehicles, their colour, and the time of day. This innovation provides a practical tool for synthesising realistic traffic images, with significant potential utility in traffic management and transportation planning applications. (Chapter 7). This contribution is documented in [147].

1.3 Structure of this Thesis

The remainder of this thesis is organised into seven chapters with the following structure. An introduction to the basic concepts and structures of deep learning is provided in Chapter 2. This chapter helps to better understand the process followed for the development of the applications in subsequent chapters. Particular emphasis is given to artificial neural networks that incorporate inductive bias for working with imagery and examining prevailing trends in image generation. **If you are already familiar with the basics of deep learning and deep neural networks, you can safely to skip this chapter.** Chapter 3 focuses on the history, functioning, and principal variants of Graph Neural Networks, which is the core paradigm studied in the present work. This chapter also outlines the benefits of GNNs in comparison to traditional artificial neural networks for dealing with relations and non-structured data.

The contributions of this thesis begin in Chapter 4, which outlines the design and creation of a model for generating cost maps that can be used for human-aware navigation. The maps are created from static data from the environment with the purpose of indicating the areas of greater disruption to humans by the robot. Chapter 5 extends the work on cost map generation by using dynamic data. For this, a new dataset with velocities and trajectories of the entities in a room is created, and several important modifications to the static model are made.

Chapter 6 describes the creation of two different models for human pose estimation from RGB sensors. This task is linked to the work on cost map generation, as it can be used to detect the position of people when generating the maps. The first model of this chapter estimates only the orientation and the torso position of the person, while the second one predicts full skeletons in 3D. Both models are suitable for multi-camera and multi-people environments.

Chapter 7 explores an application of GNNs for traffic image generation. A GNN, along with an image generation model, is employed to create realistic images of a traffic crossroad. This application demonstrates the advantages of graph neural networks in exploiting not only metric information but also semantic features.

Chapter 8 concludes this thesis, deriving insights from the experimental studies carried out in Chapters 4, 5, 6, and 7. The research questions and contributions delineated above are revisited, culminating in a discourse on the implications of these outcomes and potential future research trajectories arising from this thesis.

Chapter 2

Basics of Deep Neural Networks

As stated in the contributions, Section 1.2, most of this thesis dissertation focus on the study of Graph Neural Networks, often in combination with classic DL models. This chapter introduces the basic concepts, mathematical notation, and terminology used in the following chapters, as well as the most popular Deep Neural Networks (DNNs). The aim is to provide a theoretical and contextual framework that assists the reader in understanding the rest of the document.

Although this chapter introduces only the conventional Artificial Neural Networks (ANNs), the following chapter will be dedicated exclusively to the fundamentals of Graph Neural Networks and their main variations.

2.1 Mathematical Notation

To aid in maintaining a coherent and consistent notation throughout the document, this section provides a concise collection of the mathematical symbols and conventions for further reference. This notation follows the one in [59]:

Numbers and Arrays

s	A scalar (integer or real)
$\vec{v}$	A vector

M	A matrix
$\boldsymbol{T}$	A tensor
I_n	Identity matrix with n rows and n columns

Sets and Graphs

$\mathbb{S}$	A set
$\mathbb{R}$	The set of real numbers
$\mathcal{G}$	A graph

Indexing

v_i	Element i of vector $\vec{v}$, with indexing starting at 1
$M_{i,j}$	Element i, j of matrix M
$\boldsymbol{T}_{i,j}$	Element i, j of tensor $\boldsymbol{T}$
$j \in \mathcal{N}(i)$	Indices j of nodes connected to node i
$\mathcal{N}^n(i)$	n^{th}-order neighbors of i, including i

Linear Algebra Operations

$a = b$	a is equal to b
$a := b$	The value of b is assigned to variable a
$\vec{a} \parallel \vec{b}$	Concatenation of vector $\vec{a}$ with vector $\vec{b}$
M^T	Transpose of matrix M
$\boldsymbol{T}^T$	Transpose of tensor $\boldsymbol{T}$
$A \odot B$	Element-wise (Hadamard) product of A and B
$A \times B$	Matrix multiplication of A and B

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_x y$	Gradient of y with respect to $\vec{x}$

$\nabla_X y$	Matrix derivatives of y with respect to X
$\nabla_{\boldsymbol{X}} y$	Tensor containing derivatives of y with respect to $\boldsymbol{X}$

Probability

$\mathbb{P}(x)$	Probability distribution of a discrete random variable x
$\mathbb{P}(x \mid y)$	Probability distribution of a discrete random variable x, given y
$p(x)$	Probability distribution of a continuous random variable x
$x \sim \mathbb{P}(x)$	Random variable x sampled from $\mathbb{P}(x)$
$\mathbb{E}_{p\sim\mathbb{P}(x)}[f(x)]$	The expected value of $f(x)$ given that x is sampled from $\mathbb{P}(x)$
$\mathcal{U}(a, b)$	The uniform real distribution on the range $[a, b]$
$\mathcal{N}(\mu, \sigma)$	The normal distribution with mean μ and standard deviation σ

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function or operator f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(x; \theta)$	A function f of x parameterised by θ
$\boldsymbol{log}\ x$	Natural logarithm of x
$\|x\|_p$	L^p norm of x
$\|x\|$	L^2 norm of x

2.2 Basics of Machine Learning, the Multi-Layer Perceptron

This first section will guide the reader through the basic concept of DNNs using the Multi-Layer Perceptron (MLP) as the explanation baseline. MLP's simplicity makes it easier to understand the structure and elementary processes within Artificial Neural Networks (ANN) and the definitions here can be extrapolated to more complex DNNs. This section does not intend to provide an exhaustive explanation of the training and optimisation of processes but to introduce the key ideas that will help understand the rest of the thesis.

$\surd$ The ***Multi-layer Perceptron (MLP)*** was one of the first ANNs to appear in the machine learning community. It is an extension of the Perceptron developed by Rosenblatt in 1958 [153]. This kind of network, in theory, can "learn" to approximate any kind of linear or non-linear functions. Thereupon, we will dip into the MLP structure to understand how it computes its outputs.

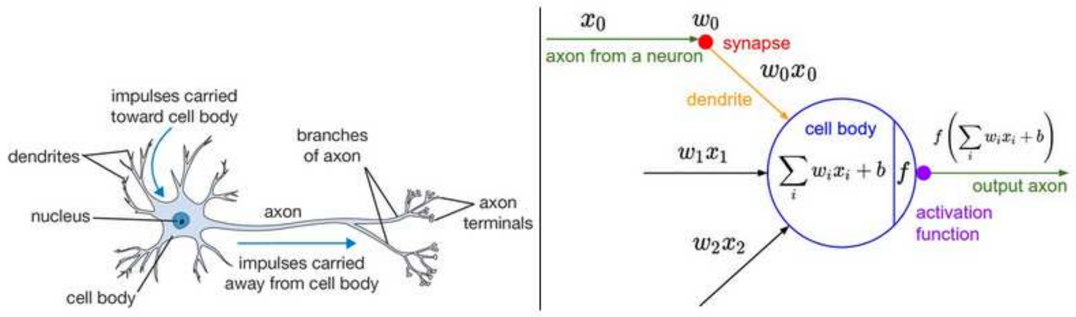


Figure 2.1: A drawing of a biological neuron (left) and the basic unit of an MLP the Perceptron (right)[1].

√ The building block of the MLP is the ***Perceptron*** referred to as an artificial neuron. It is inspired by biological neurons, sharing some similarities as depicted in Figure 2.1. It aggregates information from different inputs (dendrites) and generates a single output (axon) that is triggered using an activation function.

$$a = \sigma(\sum_i w_i x_i + b) \tag{2.1}$$

Equation 2.1 describes how the Perceptron calculates its output. The values x_i from the different i inputs are aggregated by performing a weighted sum across all inputs multiplied by some weights w_i and adding a bias term b. Therefore, a single Perceptron can approximate linear functions. It is worth noting that the bias term allows the Perceptron to learn functions that don't necessarily have to pass through the origin. As the reader might have noticed, this is the same equation as a logistic regression where the weights play the same role as the coefficient in the regression.

Although the Perceptron by itself is a good approximator for simple linear functions, the real power of the MLP comes from the connection in layers of many of these neurons forming a network. Moreover, the result of Equation 2.1 is passed through a non-linear operation √ $\sigma(\cdot)$, also called ***activation function***, that modules the intensity and range of the values at the output, thereby enabling the MLP to approximate non-linear functions. Further

[1]Source: `https://cs231n.github.io/neural-networks-1/`

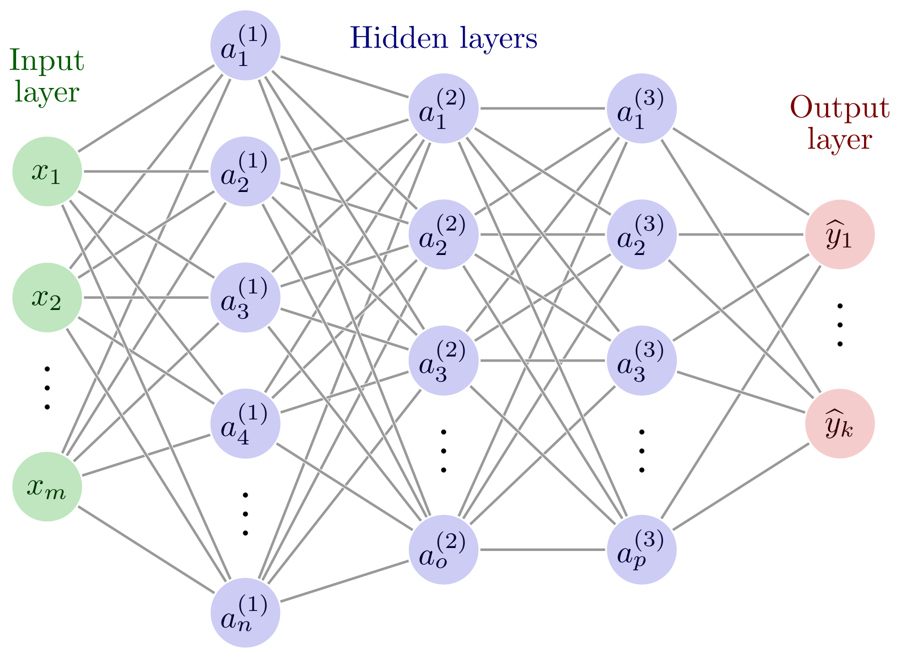


Figure 2.2: Multi-Layer Perceptron scheme.

information on these activation functions and the most commonly employed variants can be found in Section 2.2.1.

Figure 2.2 shows how several Perceptrons are connected to form the Multi-Layer Perceptron. Each row of neurons is called a ***layer*** and networks can have different numbers of layers. All the outputs of the neurons of the previous layer are connected to the inputs of each of the neurons of the second layer, which is why MLPs are also known as fully connected networks. Conventionally, these layers are separated into three parts as shown in Figure 2.2: the input layer, the hidden layers and the output layers. We see that an ANN is nothing more than a parametrised composition of affine and non-linear functions $G(\theta; x) = y$ where $\theta_i = \{W_i, b_i\}$.

Historically, if there is more than one hidden layer the ANN is considered a ***Deep Neural Network (DNN)***. According to the universal approximation theorem proved by Cybenko [43], a neural network with only one hidden layer and sigmoid-like activation functions can approximate any real function. Then why add more than one layer? The answer to this question is not clear from a theoretical point of view since there aren't convincingly demonstrations of the possible explanations. However, experience tells us that deeper ANNs lead to better results, generalising better over unseen input data [59].

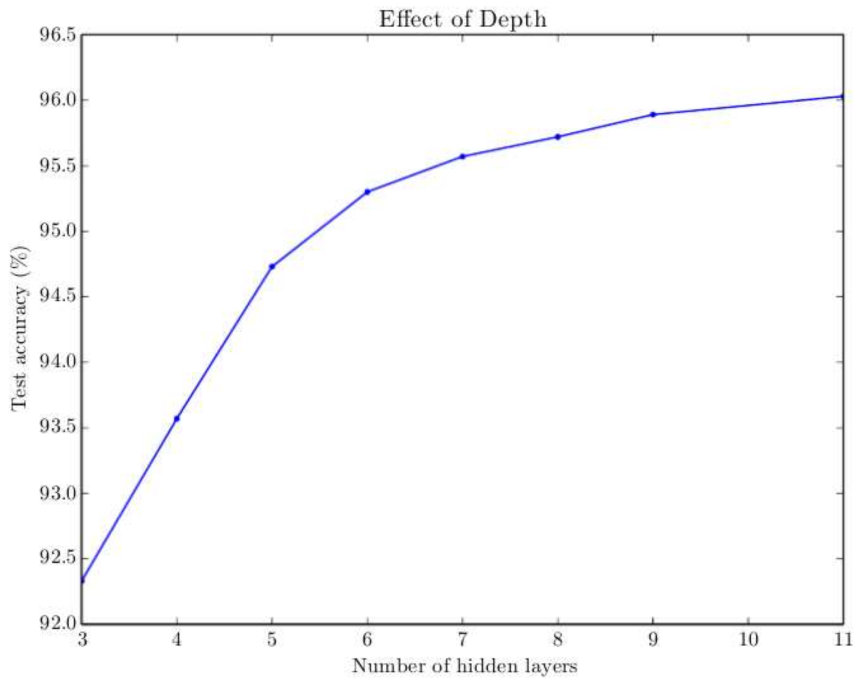


Figure 2.3: Relation between the number of layers and accuracy of a model trained to transcribe multi-digit numbers from photographs of addresses by [59].

The MLP can learn how to approximate functions after a training process where numerous pairs of inputs and expected outputs are presented to the network. The set of all these √ pairs used for training the network is called ***training dataset*** or training set. During the training process, the weights (or parameters) of the network are updated to minimise the error between the predicted output and the expected output. The most widely known and √ used mechanism to perform this training process is called ***stochastic gradient descent***, explained in more detail in section 2.3.

Next, let us see how the MLP performs a forward pass using vectorised notation. The input layer (most left-hand side in Figure 2.2) takes the data to be processed by the network. We can express the input values of the network as a vector $\vec{x} = (x_0, x_1 ... x_m)^T$. We can now transform the equation of the computation of a single Perceptron (Equation 2.1) in its vectorised version:

$$
\begin{aligned}
z_i^{(l)} &= \vec{w}_i^{(l)T} \vec{x} + b_i^{(l)} \\
a_i^{(l)} &= \sigma(z_i^{(l)})
\end{aligned}
\tag{2.2}
$$

The subscript i refers to the neuron's index in the layer and the superscript l to the index of the layer. The weights vector $\vec{w}$ must have the same length as the input vector $\vec{x}$ (output of the previous layer) and the final activation value a is yielded by the activation function $\sigma(\cdot)$. The equation of the computation for the whole layer can be written as follows in the vectorised form:

$$\begin{aligned} \vec{z^{(l)}} &= W^{(l)}\vec{x} + \vec{b}^{(l)} \\ \vec{a^{(l)}} &= \sigma(\vec{z^{(l)}}) \end{aligned} \tag{2.3}$$

Where W is now a two-dimensional matrix with the number of rows equal to the number of neurons in layer l and the number of rows equal to the number of neurons in layer $l-1$, and $\vec{b}$ is a vector of the same size as the neurons in the layer l. Finally, if we want to do a forward pass through a dataset with m samples we can write Equation 2.4 as:

$$\begin{aligned} Z^{(l)} &= W^{(l)}X + B^{(l)} \\ A^{(l)} &= \sigma(Z^{(l)}) \end{aligned} \tag{2.4}$$

In this case, B, Z and A are matrices of dimension (n_l, m), W has dimensions (m, n_l, n_{l-1}) and the input matrix X has dimensions (n_{l-1}, m). Where n_{l-1} is the dimension of the input and n_l is the dimension of the output for that layer.

When these equations are translated into code run by a program, using the vectorised version instead of loops makes it run much faster. The vector operations can be parallelised taking advantage of the graphic card's power.

2.2.1 Activation Functions

The activation function regulates when and with what intensity the neuron fires. These functions are essential for allowing the network to approximate non-linearities. In the case of not using a non-linear activation, the Multi-layer Perceptron would be nothing but a succession of linear operations that results in a linear function approximator.

It is worth noting that the activation functions of the hidden layers and the one of

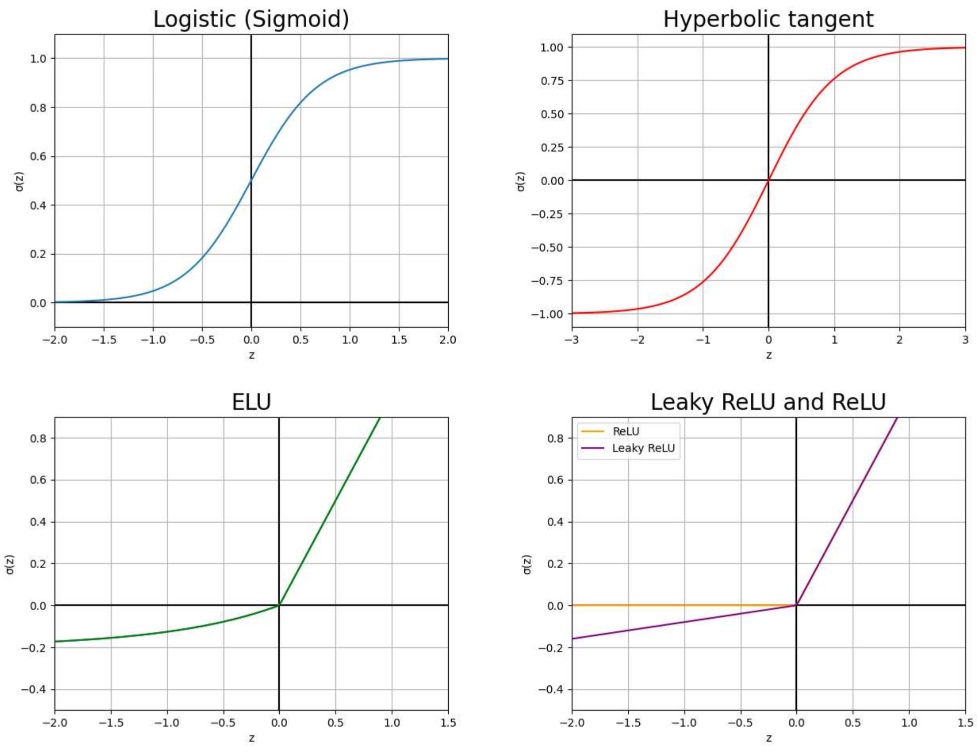


Figure 2.4: Different plots of the most relevant activation functions for this thesis.

the output layer can be different. Depending on the requirements of the network and the expected output, can be beneficial to use one function instead of another. This section goes through the activation functions explaining their pros and cons and the criteria for using them in the models in this thesis. Figure 2.4 shows the graph for each of the activation functions listed below:

- The family of **Sigmoid** functions are also used in logistic regression. Their mean is 0.5 and they only return positive numbers since their outputs go in the range $[0, 1]$. They usually perform worse than the rest of the activation functions in this list as activation for the hidden layers [132]. For this reason, in this thesis, they have only been used as activation of the output layer in the cases of binary classification or regression when the values are expected to be in the range $[0, 1]$.

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{2.5}$$

- The **Hyperbolic tangent** is a shifted and stretched version of the sigmoid function where the output values are in the range $[-1, 1]$, having a mean for the outputs in zero. In most applications, it works better than the sigmoid function for the hidden layers since convergence is faster if the average of the input of each layer is close to zero, reducing the bias shift effect [106].

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.6}$$

- The **Rectified Linear Unit** or **ReLU** [58] is one of the most popular activations for DNNs. The derivative of the points of this function on the positive side is greater in comparison with the sigmoid and the hyperbolic tangent making the network learn faster in most cases. The simplicity of its derivative makes it computationally efficient and it is a non-saturating activation function preventing gradient vanishing, which is a problem when training the network as seen in Section 2.3.

$$ReLU(z) = \boldsymbol{max}(0, z) \tag{2.7}$$

- The **Leaky Rectified Linear Unit** or **Leaky ReLU** [116] is a type of activation function based on a ReLU, but it has a small slope for negative values instead of a flat slope. The slope coefficient is a hyperparameter that can be adjusted, with Equation 2.8 exemplifying a value of 0.1. This activation function does not suffer from a ReLU problem known as "dying ReLUs", where some neurons die out, meaning they keep on throwing 0 as outputs with the advancement in training. It also has the advantage of pushing mean unit activations closer to zero. From the insights gleaned throughout the projects undertaken in this thesis, this activation function alongside the hyperbolic tangent, usually delivers the best performance.

$$leakyReLU(z) = \boldsymbol{max}(0.1z, z) \tag{2.8}$$

- The **Exponential Linear Unit (ELU)** [40] is similar to the leaky ReLU. The difference is that the negative part is softened by a parameter α as you can see in Equation

2.9. ELUs saturate to a negative value with smaller inputs and thereby decrease the forward propagated variation and information.

$$ELU(z) = \begin{cases} z & z \geq 0 \\ \alpha(e^z - 1) & z < 0 \end{cases} \tag{2.9}$$

It is worth noting that if we want the network output to be in the range of the real numbers $x \in \mathbb{R}$, it is possible to not use any activation function in the output layer or it can also be called **linear activation function**.

2.3 How the Multi-Layer Perceptron Learns

The learning process of a DNN is formulated as an optimisation problem. It is an iterative method where the parameters of the network are updated to better approximate the function matching the input space to the output space of the dataset. To evaluate how good the prediction of the network is in comparison to the expected value in the training dataset, a

√ ***loss function*** is used to provide a score for this approximation. The lower the score, the better the prediction by the network. Therefore, the learning method aims to obtain the parameter values that minimise the result of the loss function, or in other words, find the global minimum of the loss function. In reality, in a multidimensional problem, it is almost impossible to reach this global minimum and reaching a local minimum is a good enough result for the neural network training. The optimisation process consists in backpropagating the gradient of the loss function and updating the weights in the opposite direction of the gradient. This process is explained in the next section.

2.3.1 Gradient Descent

As an intuition on how gradient descent works, we can think of a one-dimensional loss function, for example, the quadratic function $\mathcal{L} = x\theta^2$. Where θ represents the single parameter of the network, and x is the input that will have the value one for this example. Evaluating the function derivative for any value of θ gives the function slope at that point, or in other words, the change rate of the function. The sign of the derivative also tells us the change in direction. If the function is increasing, the derivative will be positive and vice-versa. If we move the parameter θ in the direction of the decreasing function, an amount proportional to its derivative value, we will eventually reach the local minimum. Figure 2.5 visually exemplifies this optimisation process.

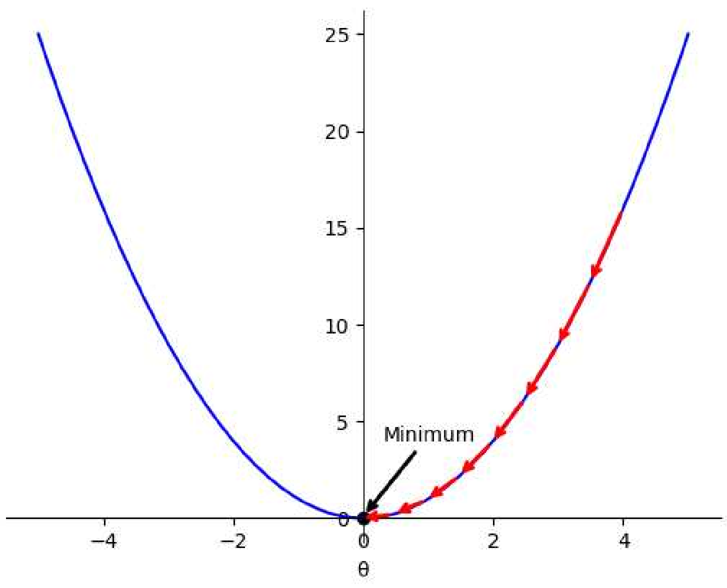


Figure 2.5: Intuition for gradient descent with one-dimensional function. The red arrows indicate the leaps during the optimisation process, leading to the global minimum.

Each leap in the figure represents an update in the θ value calculated in the following manner:

$$\theta := \theta - \alpha \frac{d\mathcal{L}}{d\theta} \; ; \; where \; \frac{d\mathcal{L}}{d\theta} = 2\theta \tag{2.10}$$

√ The *alpha* value in Equation 2.10 is called ***learning rate***, and it regulates the amount of the update of the parameters in the network. An extremely low learning rate will make the learning process slow, while a high value can cause the network training to be unstable and fail to converge. Typical values for the learning rate are lower than 1 and greater than $1e-6$ [19].

In the previous paragraphs, we have seen an intuition of how gradient descent works for a network with a single parameter. Real networks can have thousands of parameters,

increasing the dimensionality of the problem. However, the same principles can be applied, in this case calculating the gradient instead of the derivatives and evaluating the network with a batch of data despite a single point. When the loss function is calculated for a batch
√ of data it is called ***cost function*** ($\mathcal{J}$) by convention. Here, the two cost functions used in the projects of this thesis are explained depending on the application. For regression
√ problems, the ***mean square error (MSE)*** is the usual way to go:

$$\mathcal{J}_{MSE} = \frac{1}{m} \sum_{m}^{i=1} \left\| \vec{y_i} - \vec{\hat{y_i}} \right\| \tag{2.11}$$

On the other hand, for classification problems, the output of the network is passed by a
√ ***softmax*** (Equation 2.12) layer that calculates the normalised vector of classes probabilities
√ and then we apply the ***cross entropy loss (CE)*** (Equation 2.13) function. This function gives sparse values for different output classes that have similar inputs, which is the main reason why it is preferred over the MSE for classification problems.

$$\mathbb{P}(\vec{x} \in class\ j) = \hat{y}_{ij} = \frac{\boldsymbol{exp}(z_{ij})}{\sum_{k=1}^{K} \boldsymbol{exp}(z_{ik})} \tag{2.12}$$

$$\mathcal{J}_{CE} = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{K} y_{ij} \boldsymbol{log}(\hat{y}_{ij}) \tag{2.13}$$

Both the CE and MSE loss functions have smooth gradients that are simple to compute, thereby facilitating the training process and its convergence.

As mentioned, the optimization algorithm computes the gradient of the cost function updating the parameters of the network in the process. Continuing with the MLP example, the first step is to do a forward pass to calculate the outputs for a set of inputs following the Equations 2.4 in Section 2.2. Then the result of the loss function is calculated using the outputs of the network and the expected outputs in the dataset. Using the chain rule the gradients are calculated as follows from the cost function:

$$\begin{aligned}
\nabla_{\boldsymbol{Z}}\mathcal{J}^{(l)} &= \nabla_{\boldsymbol{A}}\mathcal{J}^{(l)} \odot \sigma^{(l)}(\boldsymbol{Z}^{(l)}) \\
\nabla_{\boldsymbol{W}}\mathcal{J}^{(l)} &= \frac{1}{m}\nabla_{\boldsymbol{Z}}\mathcal{J}^{(l)}\boldsymbol{A}^{(l-1)} \\
\nabla_{\boldsymbol{B}}\mathcal{J}^{(l)} &= \frac{1}{m}\sum^{m}\nabla_{\boldsymbol{Z}}\mathcal{J}^{(l)} \\
\nabla_{\boldsymbol{A}}\mathcal{J}^{(l-1)} &= \boldsymbol{W}^{(l)T}\nabla_{\boldsymbol{Z}}\mathcal{J}^{(l)}
\end{aligned} \tag{2.14}$$

And the parameters of each layer are updated as follows:

$$\begin{aligned}
\boldsymbol{W}^{(l)} &:= \boldsymbol{W}^{(l)} - \alpha\nabla_{\boldsymbol{W}}\mathcal{J}^{(l)} \\
\boldsymbol{B}^{(l)} &:= \boldsymbol{B}^{(l)} - \alpha\nabla_{\boldsymbol{B}}\mathcal{J}^{(l)}
\end{aligned} \tag{2.15}$$

In the example explained above, we have assumed that the dataset contains the expected outputs for the corresponding inputs and the training process is called ***supervised learning***. Most models developed during this thesis have used supervised learning for the training. However, due to the difficulty of gathering data for some specific applications, we often do not have labels (or expected outputs) in our dataset. For instance, other learning approaches such as self-supervised learning was explored in Chapter 6.

2.3.2 Initialisation and Normalisation

Before plunging into the optimisation of gradient descent, it is crucial to underscore the importance of correctly initialising the network weights and normalising the input. Regarding the ***initialisation*** of the learnable parameters, one possibility is to set all of them to the same value. However, this is not a favourable option, as it results in all neurons performing identical operations and processing the same gradients, therefore leading to the so-called ***symmetry problem***. In such a configuration, the network will not be able to learn and update its parameters, making it useless to have more than one neuron per layer. For this reason, the initial weight values are sampled from a random distribution. However, the choice of the distribution must be done carefully to avoid vanishing/exploding gradient problems or falling into the aforementioned symmetry problem. Gradients vanishing become too small to have any effect and exploding become too large, leading to numerical instability. The selection of the distribution is often based on the number of inputs or fan in, the number of outputs or fan out, the type of activation and the type of network. The three most common initialisation techniques in deep learning are listed here:

- The LeCun initialisation [107] normalises the variance of the sampling distribution to avoid it growing with the number of inputs. This allows the neurons have a significant output variance. The weights are drawn i.i.d. with zero mean and a normalised variance with the number of inputs n_{in}:

$$\boldsymbol{W} \sim \mathcal{N}(0,k) \ where \ k = \frac{1}{n_{in}} \tag{2.16}$$

- Xavier initialisation [57] also takes into account the fan out n_{out} for a more efficient performance during backpropagation. It works better with sigmoid and hyperbolic tangent activations.

$$\boldsymbol{W} \sim \mathcal{N}(0,k) \ where \ k = \frac{2}{n_{in} + n_{out}} \tag{2.17}$$

- Finally, He initialisation [67] introduces a slight modification and it works better with ReLU or LeakyReLU activation types.

$$\boldsymbol{W} \sim \mathcal{N}(0,k) \ where \ k = \frac{2}{n_{in}} \tag{2.18}$$

In practice, all the frameworks utilised throughout this thesis offer a default initialisation for each type of layer. Based on my experience, employing these default initialisations has proven effective, and I have not encountered the need to alter or fine-tune them. For instance, the default initialisation for the weights of a fully connected layer in PyTorch [2] is:

$$\boldsymbol{W} \sim \mathcal{U}(-\sqrt{k}, \sqrt{k}) \ where \ k = \frac{1}{n_{in}} \tag{2.19}$$

[2]Documentation for a fully connected layer in the framework PyTorch: `https://pytorch.org/docs/stable/generated/torch.nn.Linear.html`

√ Turning now to the ***normalisation*** topic, it is also a crucial step before training the network. The normalisation step sets the inputs to have zero mean and the variance to 1, this way all the input variables are in a similar range. The normalisation of the input creates a more symmetric cost function, which helps to stabilise the gradient descent step, allowing us to use larger learning rates or help models converge faster for a given learning rate. Figure 2.6 depicts an intuition of how the normalisation can affect the shape of the cost function for a hypothetical network with two parameters. Normalising the input data can also help the model to generalise better to new data. This is because the model is less sensitive to the scale of the input data, and is, therefore, less likely to overfit the training data. Normalised inputs also prevent the problem of exploding and vanishing gradients.

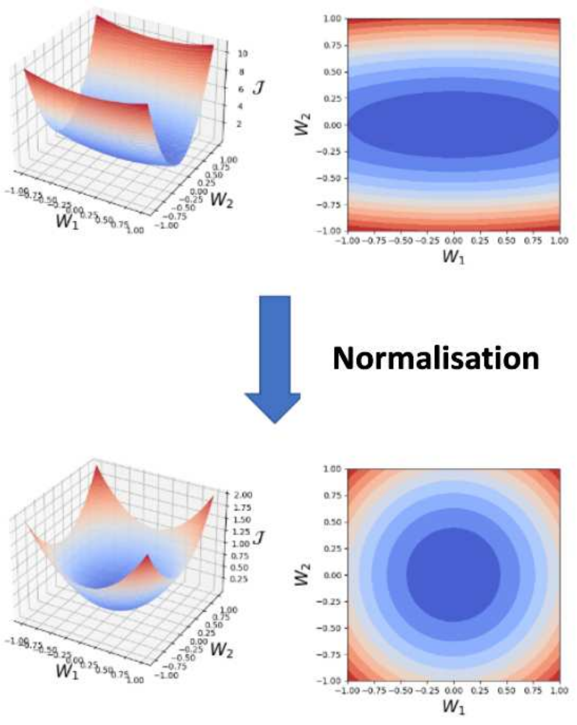


Figure 2.6: An illustrative example of how the normalisation of the input can make the loss function more symmetric, which speeds up and stabilise the learning process.

√ It is important to include in this section the ***batch normalisation*** technique [79]. It follows similar principles of normalising the input data but in this case, it normalises the activations $\vec{z}$ of the previous layer at each batch before the activation function is applied by subtracting the batch mean μ and dividing by the batch standard deviation $\vec{\sigma}^2$. Equations 2.20 show how the activation vector is normalised across the length of the batch m.

$$\vec{\mu} = \frac{1}{m}\sum_{i=1}^{m}\vec{z}_i \qquad \vec{\sigma}^2 = \frac{1}{m}\sum_{i=1}^{m}(\vec{z}_i - \vec{\mu})^2 \qquad \vec{z}^i_{norm} = \frac{\vec{z^i} - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2.20}$$

After the activation vector is normalised the activations are updated as follows:

$$\vec{\tilde{z}^i} = \gamma \vec{z}^i_{norm} + \beta \tag{2.21}$$

Where γ and β are learnable parameters, ϵ is a constant added for numerical stability. This has the effect of stabilising the distribution of activations and reducing the covariate shift, which is the change in the distribution of inputs to a layer caused by the change in the distribution of outputs from the previous layer.

There are several benefits to using batch normalisation:

1. Batch normalisation has been shown to significantly improve the performance of neural networks, especially when the network is deep or has a lot of layers [79]. It speeds up and stabilises the training following the same intuition seen earlier with the normalisation of the input data.

2. In some cases, batch normalisation can reduce the need for dropout, a regularisation technique that we will see Section 2.3.5 used to prevent overfitting since it also adds a slight regularisation effect.

3. Batch normalisation can make it easier to tune the hyperparameters of a neural network, such as the learning rate because it helps to stabilise the activations and gradients. This can make the network more robust to changes in the hyperparameters and allow for more efficient training.

2.3.3 Optimisation

We have seen thus far how a neural network learns using gradient descent by processing the whole dataset with a forward pass and computing the gradient to update the parameters. This section points out several techniques to improve the learning process.

$\surd$ The first technique is called the ***mini-batch gradient descent***. As mentioned in previous sections, it is a common practice to divide the training dataset into smaller sets called mini-batches of customisable size (tuneable hyperparameter). If this size is just one training $\surd$ sample the algorithm is called ***stocastic gradient descent***. When the forward pass and posterior backpropagation of the gradients is done with fewer data, the parameters of the network are updated more often and therefore, the learning process is faster. For this reason, mini-batches gradient descent is especially advantageous for training with big datasets, which is the common trend in deep learning.

When the network has processed all the mini-batches in the dataset, we can say that the $\surd$ algorithm has run through an ***epoch***. The number of epochs is also a hyperparameter of the network that can be modified depending on the convergence speed of the specific network. It is worth mentioning that the order of the mini-batched in the dataset is randomised in each epoch since it usually leads to faster convergence by avoiding local minima [19].

The primary drawback of mini-batch gradient descent is that the steps taken in each iteration deviate more from the local minima compared to those taken with the entire dataset. This occurs because the parameter updates are computed using fewer data points. The smaller the mini-batch size, the more the step diverges from its path towards the local minima of the cost function. Consequently, the choice of batch size represents a trade-off between computational speed and the convergence of the algorithm. In practice, the memory of the GPU can also limit the batch size. The black arrows in Figure 2.7 represent the updates with mini-batch gradient descent in a simplified network with only two parameters.

As the reader can observe in Figure 2.7, the mini-batch gradient descent is sensitive to the non-symmetries in the cost function. In this example, the steps in the direction of the parameter W_2 are much larger compared to the ones in W_1 direction. This is because W_2 changes faster and therefore its derivatives are bigger, resulting in larger updates. This non-symmetry problem can be solved by adding momentum to the update of the parameters based on the values of the previous updates. There are several optimisers which add momentum to the gradient descent, yet the most popular one to the date of this thesis is $\surd$ the ***Adam optimiser*** [87]. The Adam algorithm first calculates the weighted averages of

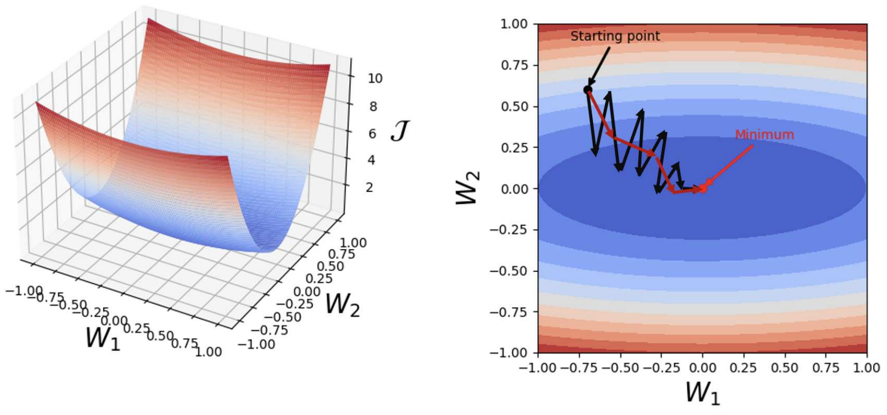


Figure 2.7: Cost function of a network with only two parameters W_1 and W_2. In the right figure, the black arrows represent the steps taken by a mini-batch gradient descent algorithm while the red arrows add momentum to the updates.

the gradients and the square of the gradients:

$$v^t_{d\theta} = \beta_1 v^{t-1}_{d\theta} + (1 - \beta_1)\nabla_{\boldsymbol{\theta}}\mathcal{J} \qquad s^t_{d\theta} = \beta_2 s^{t-1}_{d\theta} + (1 - \beta_2)\nabla_{\boldsymbol{\theta}}\mathcal{J}^2 \tag{2.22}$$

The authors recommend the values $\beta_1 = 0.9$ $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. Then a bias correction is applied to the terms calculated in Equation 2.22:

$$v^c_{d\theta} := \frac{v_{d\theta}}{(1 - \beta^t_1)} \qquad s^c_{d\theta} := \frac{s_{d\theta}}{(1 - \beta^t_2)} \tag{2.23}$$

Finally, the parameters of the networks are updated in the following way:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \frac{v^c_{d\theta}}{\sqrt{s^c_{d\theta}} + \epsilon} \tag{2.24}$$

Applying these changes to the update function makes the updates look more like the red arrows in the right plot of Figure 2.7. As you can see, there is a significant improvement in the number of steps needed to reach the minimum.

√ Another optimisation trick is called ***learning rate decay***. Normally, a large learning rate is acceptable at the start of the optimisation to speed up the process. However, when the parameters get closer to the minimum it is better to reduce the length of the steps to guarantee stability. Hence, learning rate decay decrements the size of the update step - learning rate - when the number of epochs increases. There are numerous forms of implementing this, here two common examples are shown:

$$\alpha = \frac{1}{1 + \gamma \times epoch}\alpha_0 \qquad \alpha = \alpha_0 * \gamma^{epoch} \tag{2.25}$$

A high dimensional problem is very unlikely to have local minima because all the parameters must have derivatives equal to zero at the same point for that to happen. It is more likely to end up in saddle points, forming plateaus with small gradients in specific directions that slow down the learning process. The optimisation algorithms seen in this section help avoid getting stuck in these plateaus. All the projects of this thesis use the Adam optimiser adding the learning rate decay when necessary to speed up the training.

2.3.4 Bias versus Variance

It is important to dedicate a section to the problem of "bias versus variance" as it represents
the underlying tension when training a neural network. This section starts by defining
√ both terms and then explains the close relationship between them. ***Bias*** refers to the
discrepancy between the average prediction of a model and the actual value that the model
is attempting to predict. Models with high bias tend to not fully adapt to the training data
√ and oversimplify the model. On the other hand, ***Variance*** refers to the variability in the
predictions of a model for a given data point, or the spread of the data. Models with high
variance pay close attention to the training data and do not generalise well to unseen data.

Figure 2.8 depicts three different plots representing the approximation of three models.
The red points are the data in the dataset and the blue line is the approximation of that
√ specific model. The righthand plot shows a model with a high bias that ***underfits*** the
√ data. The model represented in the middle plot, on the contrary, ***overfits*** the dataset by
capturing the noise along with the underlying pattern in the data. The goal of the training
process is to get a model like the one represented in the lefthand plot, with a good balance
between bias and variance. A model capable of capturing the complex patterns in the

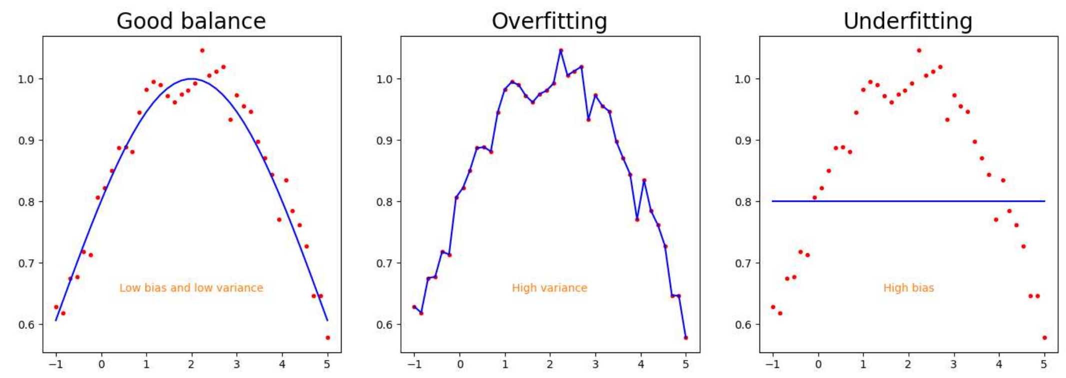


Figure 2.8: Difference between bias and variance and their relation to overfitting and underfitting the training data.

dataset and, at the same time, capable of extrapolating its predictions to unseen data, not getting lost in the noise.

There are several ways of checking if our model is overfitting or underfitting the training data. One of the most common practices is to split the dataset into three subsets: the training set, the development or validation set and the test set. The training process involves multiple cycles, known as epochs, where the model goes through the entire training dataset. After a predetermined number of iterations, the model performs a single forward pass on the validation dataset to assess its performance. Once the training process concludes, the final model's performance is evaluated using the test dataset. Normally, for big datasets, the proportion of the training set is much bigger than the dev and test sets. For example, the split proportions could be 0.9, 0.05 and 0.05 for training, dev and test sets respectively. Thus, if the error over the training set is low and the error over the validation set is relatively higher, it is probable that the model has a high variance. If the error is high in both sets, the model probably has a high bias. On the other hand, a high training error and even higher validation error can mean high variance and bias (usually due to small datasets). Determining what constitutes a high training error is not always straightforward and is often assessed by the researcher based on their expertise in the field of knowledge. It is worth noting that the development and test sets must not have any samples from the training √ set. This is considered ***data leakage*** and affects the evaluation of the model. Generally, data leakage occurs when information from outside the training dataset is utilised to create the model. Additionally, data leakage may also be considered when dealing with extremely similar data points, such as consecutive frames in a video. The test set is employed to compare performances across various models trained with different hyperparameters. We will see this in more detail in Section 2.3.6.

We have seen how to detect if our model has a high variance or a high bias, but how can we correct these problems to obtain a balanced network? As a general rule of thumb, bigger networks or longer training can fix the problem of high bias but also increase the risk of high variance. On the other hand, more data and regularisation techniques can solve the problem of high variance but risk the high bias again. The most common regularization techniques to avoid overfitting (high variance) will be explained in Section 2.3.5. As can be deduced, achieving a model with low bias and low variance is an iterative give-and-take problem.

To conclude this section, it is worth mentioning another training strategy known as k-fold cross-validation. This method involves dividing the training set into k parts and conducting k training iterations, each time using a different portion as the validation set [129]. Since this strategy was not explored in this thesis, no further explanation of this technique is provided in this document.

2.3.5 Regularisation

As previously mentioned, there are two main ways of preventing overfitting and decreasing the bias component of the model: training with more data and regularisation techniques. Since obtaining more training data is often problematic, it is crucial to know how to use regularisation. In this section, we will explore the most prevalent regularisation techniques in deep learning.

√ To begin, we will discuss ***L2 regularisation***, a technique that involves reducing the weights of the network during the backpropagation of the gradient. This method is also known as weight decay. To implement L2, the cost function is modified by adding the following regularisation term:

$$\mathcal{J}(W,B) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^i, y^i) + \frac{\lambda}{2m}\|W\| \tag{2.26}$$

The regularisation hyperparameter λ controls the strength of regularisation. As λ increases, the regularisation effect becomes more pronounced. In practice, the L2 regularisation is implemented in the optimiser of most Python frameworks. For example, for PyTorch, the Adam optimiser class accept a weight decay parameter to implement this regularisation [3].

√ ***Dropout*** [167] is another effective regularisation technique that involves randomly deactivating a certain proportion of neurons in the network during each forward pass of the algorithm. This is achieved by setting a probability for each layer to suppress a neuron. The weights of the inactive neurons are not updated during the backpropagation of the gradients.

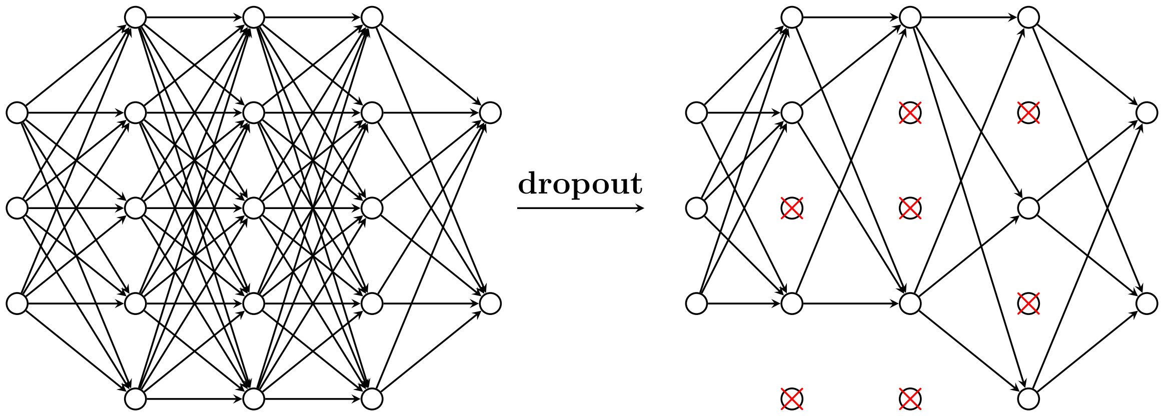


Figure 2.9: Representation of how the dropout technique deactivate a random number of neurons in the hidden layers. Note that the connections of this neurons are also removed.

As shown in Figure 2.9, deactivating a neuron through dropout means also removing its input and output connections, so it does not affect the network. This effectively causes each update to a layer during training to be performed with a different "view" of the configured layer, adding noise to the training process. The resulting architecture forces nodes within a layer to probabilistically take on more or less responsibility for the inputs. This interpretation suggests that dropout may disrupt cases where network layers co-adapt to correct errors from previous layers, thereby making the model more robust.

√ Another interesting and useful technique is called ***data augmentation***. As previously mentioned, having more data for training is always beneficial and improves the problem

[3]Documentation for Adam optimiser in PyTorch framework: `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`

of high variance. However, it is often costly and hard to gather more data points for our dataset. Data augmentation consists of applying transformations to the available data to create new data points. These transformations must yield data in the same distribution of the dataset. For example, data augmentation is a common practice in image classification where images in the training dataset can be flipped, cropped or rotated to get new data.

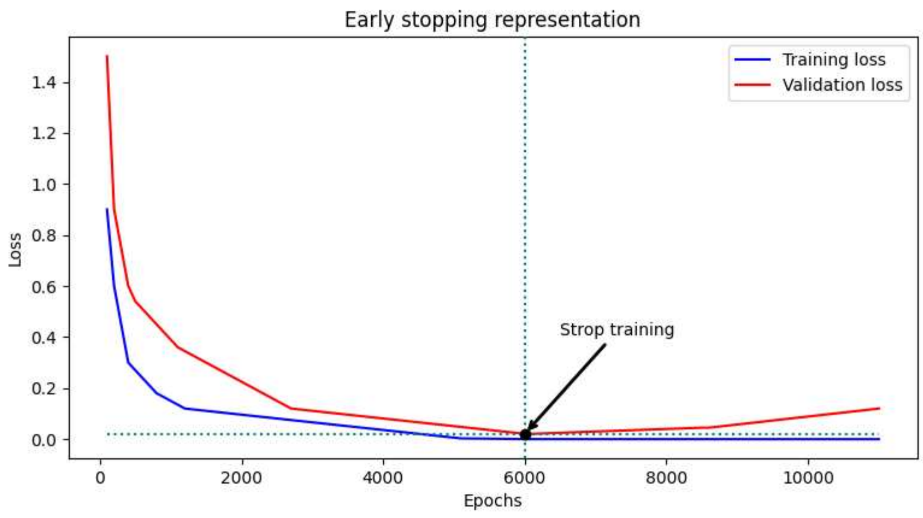


Figure 2.10: Simulation of the training and validation loss during training time. The intersection of the dotted lines marks the optimal point where the training should stop to avoid overfitting.

√ Finally, ***early stopping*** is a technique that involves terminating the training loop when the error on the development set has not improved after a certain number of epochs, in √ order to prevent overfitting. This number is called ***patience*** and is a hyperparameter that can be adjusted. The training aims to perform well on unseen data. Therefore, when the validation error starts to increase, it indicates that the network is overfitting the training data and is no longer improving its performance on new data points. Figure 2.10 depicts both the training set and validation set errors and when it is optimal to stop the training to avoid overfitting.

2.3.6 Hyperparameter Tuning

√ The ***hyperparameters*** of a deep neural network are the adjustable parameters that govern the network's architecture, learning process, and optimization. Not to be confused with the learnable parameters of the network. The choice of hyperparameters can significantly impact the model's performance and convergence rate. Previous sections mentioned and explained most of the commonly used hyperparameters to define a neural network such as the number of layers and hidden units, the learning rate, the type of activation function, the batch size, the regularization technique, etc. Unfortunately, there is no way to calculate the hyperparameters to get the lowest loss, therefore, these hyperparameters are tuned in an iterative process until reaching a good result. The performance of models with different hyperparameters is evaluated over a test set which must not have data leakage as explained in Section 2.3.4. The goal of the test set is to provide an unbias estimation of the performance so it can be compared to other models.

There are several methods to sample different hyperparameters. Probably, the most $\surd$ known of these methods is the ***grid search***. The selection of different hyperparameters within a reasonable range[4] is presented in a grid. In a grid search, consecutive elements have consecutive values for the hyperparameters. Each model in the grid is trained in order until the best results are found. This method assumes that hyperparameters are equally important which is not true. For example, in most cases, a small change in the learning rate affects much more the learning process than a small change in the number of neurons in a layer. Thus, sampling each hyperparameter from a random distribution often leads to $\surd$ better results faster. This process is called ***random search*** and is the one used for all the contributions of this thesis. In some specific cases, some hyperparameters were manually fine-tuned based on the experience in previous experiments.

2.4 Deep Neural Networks for Imagery

Despite its effectiveness, the Multi-Layer Perceptron (as previously described) is primarily designed for processing flat (unstructured) data. In order to make further progress on larger-scale and more complex inputs, it is advantageous to exploit simple structures in the inputs whenever possible by adding relational inductive bias. Ideally, the goal is to achieve better results with fewer parameters and less data. This section introduces the convolutional layers that can be applied to image-like input data. Some of the contributions of this thesis explore the generation of images with neural networks and the following sections will summarise the basic concepts and architectures used for that purpose. It is worth noting

[4]A reasonable range for the hyperparameters is given by the experience of the practitioner. However, previous sections pointed out the typical range for most of them.

that the convolutional networks can be expressed as a constrained Multi-Layer Perceptron; however, an MLP would require a significantly larger amount of training data to uncover the structural inductive biases present in this architecture.

2.4.1 Convolutional Neural Networks

√

Convolutional Neural Networks (CNN) are the prime architecture for working with images in DL. The two main types of layers of CNNs -the convolutional layer and pooling layer- were first introduced in 1982 by the work of Fukushima et al., the Neocognitron [52]. Subsequently, in 1989, Lecun et al. integrated these layers with the use of gradient backpropagation for the purpose of learning convolutional kernel coefficients [105]. In recent years, advancements in computational power, particularly through the use of graphics processing units (GPUs), have facilitated the development of deeper CNNs, leading to a revolution in the field of image processing and computer vision. Notable examples of such models include Alexnet in 2012 [98] and VGG in 2014 [163], which continue to be widely used to this day. CNNs have been widely applied in various fields, including image classification, object detection, and image generation. In this text, the focus will be on the field of image generation. This is related to Chapters 4, 5, and 7, where CNNs are utilised in generating cost maps in the first two chapters and producing realistic images in the final chapter.

One of the key advantages of a convolution layer in this context is its ability to leverage two crucial relational inductive biases: parameter sharing locally and translation equivariance. Parameter sharing allows for the use of the same weight across different positions by convoluting a kernel or filter in the input image. These results in sparse connections and a reduction in the number of parameters required. Figure 2.11 shows a diagram of how a 2D convolution of a 3×3 kernel is performed. Each pixel of the output is generated by sliding the kernel over the image. In this example, the kernel is used for detecting vertical edges in an image; however, in an actual CNN, the kernel parameters are learned during the training process, and typically, multiple different kernels are implemented. Equation 2.27 describes this process in the case of an RGB channel performing a 3D convolution.

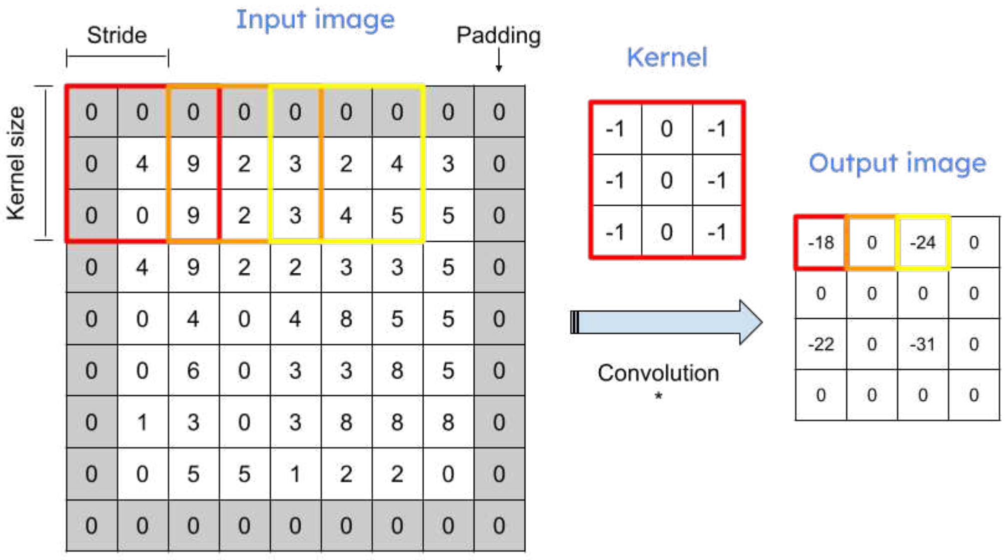


Figure 2.11: Schematic representation of a 2D convolution for vertical edges detection.

$$X'_{abc} = \sigma\left(b_c + \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{d} K_{ijkc} \cdot X_{a+i-1,b+j-1,k}\right) \tag{2.27}$$

This equation assumes an RGB image with d channels where $X' \in \mathbb{R}^{h'\times w'\times d'}$ is the result of the convolution for d' channels, $X \in \mathbb{R}^{h\times w\times d}$ is the input image, $K \in \mathbb{R}^{n\times m\times d\times d'}$ is the √ kernel and $\vec{b} \in \mathbb{R}^{d'}$ is a bias vector. ***Convolutional layers*** stack many of these layers and concatenate the results in the channel dimension of the output.

Five hyperparameters configure a convolutional layer: the kernel size, the padding, the stride, the number of kernels (or the number of channels in the output) and the number of channels in the input. Figure 2.11 shows a representation of the first three hyperparameters. The padding adds a border of zeros to the image, while the stride indicates how many pixels the kernel moves in each iteration of the convolution. These two hyperparameters help preserve spatial dimensions, prevent information loss, and control the output size of the layer. Equation 2.28 shows how the two first dimensions of the image (height and width) change in function of these parameters.

$$n_o = \left\lfloor \frac{n_i - k + 2p}{s} \right\rfloor + 1 \tag{2.28}$$

Where the output dimension n_o is calculated from the input dimension n_i, the kernel size k, the padding size p and the stride s.

√ Another essential layer in a CNN is the ***pooling layer*** that is used to increase the receptive field of neurons, and as a consequence, the images are downsampled. These layers reduce the size of the feature map, which in turn makes computation faster. They do not have any learnable parameters. These layers convolve a kernel aggregating information from the surrounding pixels with the desired operation. Probably the most used aggregation operation is the maximum and this kind of pooling layer is called max pooling.

On the contrary, there are several strategies for upsampling the input image depending
√ on the requirements. The most famous strategy is the use of ***transpose convolutional layers***. They perform the opposite operation of the convolutional layers by multiplying each pixel of the input image for each of the elements of the kernel to generate the pixels of the output image. The dimensions of the output image can be calculated as follow:

$$n_o = (n_i - 1)s - 2p + k + p_o \tag{2.29}$$

The main problem of the transposed convolutional layers is that their output tends to contain the so-called chequered pattern in the output image. This pattern arises when the kernel size is not divisible by the stride, leading to artefacts in the output image resembling a checkerboard. This issue is typically addressed by first upsampling the image using a method such as bilinear interpolation and then applying standard convolutional operations.

√ Finally, it is crucial to mention the ***residual blocks*** or skipping connections. This strategy consists in summing the activations of early layers in a CNN in later stages of the network. Residual blocks are extremely useful for deep networks avoiding the later layers "forget" information of the input. They also help prevent vanishing gradients in very deep networks.

2.4.2 Image Generation

In the previous section, we explored the basic building blocks of image processing with deep learning. This section focuses on the main architectures that build on top of CNNs for image generation.

First, we introduce the U-Net architecture. Although this network was originally designed for generating image segmentation masks, it can also be employed in the creation of simple images, such as the cost maps in Chapter 5.

Second, for generating more complex and realistic images, there are three main streams of work in the literature: Generative Adversarial Networks (GANs), Variational Auto-Encoders (VAEs), and diffusion/denoising models. Due to the ability of GANs to generate high-resolution images and their efficiency, a variant of this model is used in Chapter 7 for generating realistic traffic images. Of the three mentioned architectures, only GANs are explained in this section, as they are related to this thesis. The exploring of the promising diffusion and denoising models is proposed for future work.

U-Net

√ The ***U-Net*** architecture was initially proposed by Ronneberger et al. in their work [151] for the application of biomedical image segmentation. This architecture comprises two distinct components: a downsampling section and an upsampling section. Within the downsampling segment, convolutional and pooling layers are stacked to extract abstract representations of the input image, effectively reducing its dimensions to a latent space representation. Conversely, the upsampling section utilises transpose convolutional layers to upsample these abstract representations, restoring their spatial dimensions to match those of the original input image. Consequently, both input and output images share identical dimensions. This particular architecture is widely recognised as an encoder-decoder network.

Figure 2.12 presents a schematic illustration of the original U-Net architecture, adapted from [151]. The width of the rectangles corresponds to the channels of the volumes, while their height represents the spatial dimensions. It is important to note the presence of skip connections between the downsampling and upsampling sections, achieved by concatenating the activations from the former to the latter. These connections play a crucial role in the

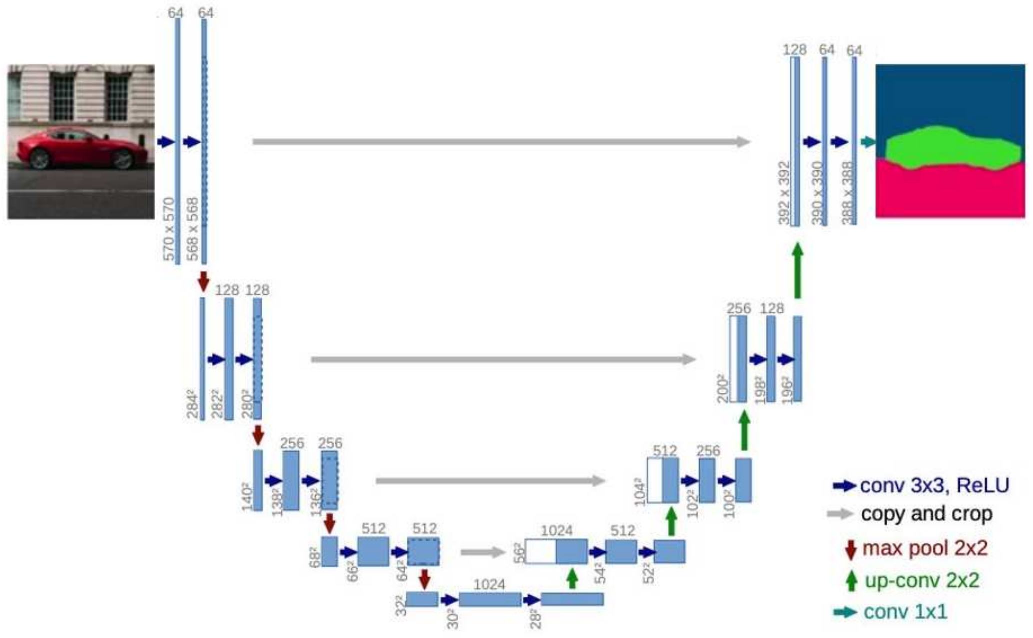


Figure 2.12: Adapted schematic representation of the U-Net architecture, based on the original design presented by Ronneberger et al. [151].

exceptional performance of the U-Net architecture. To provide an intuitive understanding of how these connections function, consider that they transmit spatial information from the network's earlier layers to its later layers. This spatial information is then combined with the abstract information extracted from the latent space, thereby enriching the output.

Generative Adversarial Network (GAN)

√ The ***Generative Adversarial Network (GAN)*** architecture was first introduced by Goodfellow et al. [61] in 2014. However, it was the remarkable outcomes achieved using deep CNNs within Deep Convolutional Generative Adversarial Networks (DCGANs) [141] that truly catapulted the popularity of GANs for image generation.

The GAN model architecture involves two sub-models: a generator model G responsible for creating new examples and a discriminator model D for classifying whether the generated examples are real, originating from the domain, or fake, produced by the generator model. The generator captures the data distribution, while the discriminator estimates the probability that a sample is derived from the training data rather than the generator, thereby functioning as a binary classifier. A key advantage of GANs is their adversarial training approach, as opposed to supervised learning, which circumvents the necessity for annotated data. In probabilistic terms, the generator learns a distribution p_g over the data x by constructing a mapping function from a prior noise distribution $p_z(z)$ to the data space as $G(z; \theta_g)$. Additionally, the discriminator, $D(x; \theta_d)$, yields a single scalar denoting the probability that x originated from the training data rather than p_g. Figure 2.13 illustrates a basic representation of a GAN's structure.

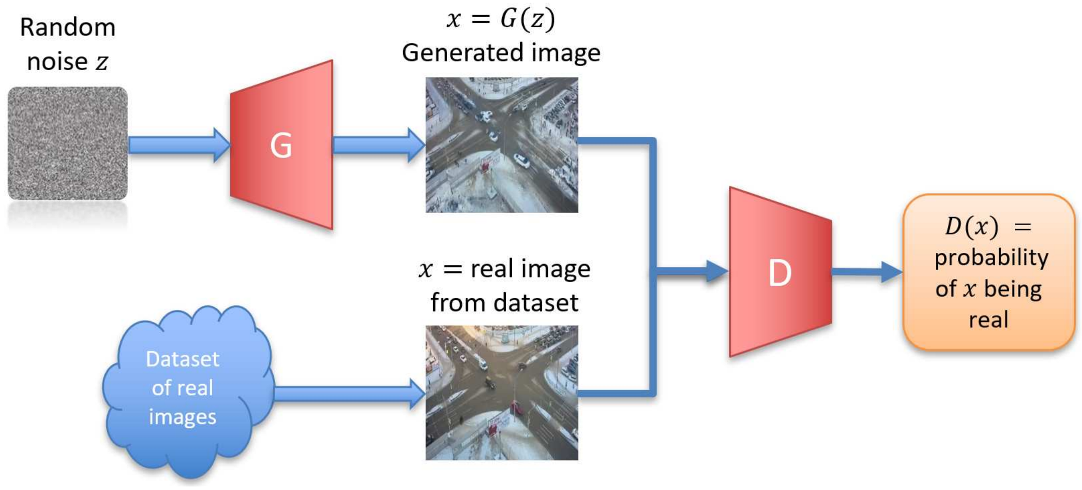

Random noise z
G
$x = G(z)$ Generated image
x = real image from dataset
Dataset of real images
D
$D(x)$ = probability of x being real

Figure 2.13: Basic schematic representation of the GAN architecture.

G and D are both trained simultaneously. We need the generator to produce fake images to train the discriminator with, and at the same time, the discriminator gives us a loss function to train the generator with (chicken and egg problem). They are trained adversarially, it is a race between two competing criteria known as a min-max game. The parameters θ_g for G are adjusted to minimise $log(1 - D(G(z)))$, and the parameters θ_d for D are adjusted to maximise $log(D(X))$ as if they are following the two-player min-max game with value function $V(G, D)$ as expressed in Equation 2.30. To facilitate convergence usually we do several updates of the discriminator parameters for each generator update.

$$\min_G \max_D V(D, G) = \mathbb{E}_{p \sim p_{data}}[\log(D(x))] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.30}$$

GANs represent a parametric approach extensively employed for image synthesis. Once trained, the generator can synthesise images based on a noise vector seed. When contrasted with the blurry and low-resolution outcomes produced by other deep learning methods such as Variational Autoencoders (VAEs) [88], GAN-based techniques [81, 136] yield more realistic results featuring higher resolution and finer details. This serves as the primary rationale for utilising this architecture in Chapter 7.

Conditional GAN

A conditional GAN (cGAN) is a modification of the standard GAN architecture that enables the conditioning of the generator's output. This concept was first introduced by Mirza and Osindero in [122], where the authors conditioned both the generator and discriminator with supplementary information **y**. This information could be any auxiliary input, such as class labels, data from other modalities, or even a graph, as proposed in Chapter 7.

Conditioning can be performed by introducing **y** to both the discriminator D and generator G as an additional input channel. In the generator, the prior input noise z, randomly sampled from a uniform Gaussian $p_z(z)$, is combined with **y** to form a joint hidden representation. The proposed objective function is given in Equation 2.31.

$$\min_G \max_D V(D,G) = \mathbb{E}_{p \sim p_{data}}[\log(D(x|y))] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z|y)))] \tag{2.31}$$

In the generator, the prior input noise $p_z(z)$ and y, the condition or constraint, are combined. The discriminator receives x and y as inputs. It is worth noting that newer versions, such as SPADE [136], introduce innovative ideas for conditioning the generator, such as embedding **y** between its layers. This will be explored in greater detail in Chapter 7.

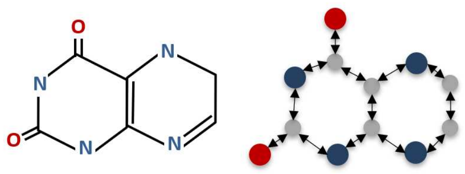
Chapter 3

Graph Neural Networks

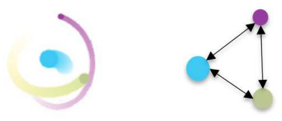
The previous chapter presents the basics of deep neural networks and the most relevant architectures with regard to the contributions of this document, leaving Graph Neural Networks (GNNs) -the main focus of this study- to this chapter. Hence, the history of GNNs, their operation, the main applications, and details of the variations used and explored in this thesis will be introduced. Throughout the discussion, the unique advantages that GNNs offer over other Deep Learning approaches for graph analysis will be emphasized, while also spotlighting the remaining limitations in the field.

√ A ***Graph Neural Network*** is a relatively novel deep learning paradigm of neural networks capable of processing graphs. According to [15], these models can generalise better over structured graph data than conventional CNN and recurrent neural networks. It induces stronger relational inductive bias into the model, leveraging the relations among data in a more efficient manner. This better performance on graphs presents a notorious advantage in real problems' usability since numerous domains are easier represented as graphs rather than images, tensors or vectors. It is worth noting that GNNs are invariant to node and edge permutations, a crucial perk in most applications as it will better explain in Section 3.3

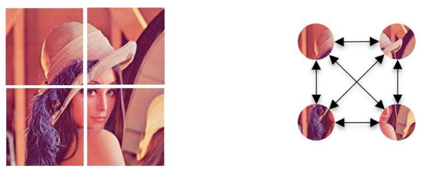
A. Sperduti et al. introduced the concept of working with structure data in graphs using neural networks in [164] in 1997. However, it was not until 2005 that the first GNN was formally proposed with the work of Marco Gori et al. [63] and first used in a real application by Franco Scarcelli et al. [157] for ranking web pages with outstanding results. The reader can find multiple surveys in GNNs, for example, [195] and [204], that show a summary of the most significant developments carried out in this field. Battaglia et al. [15] perform a review and unification of the different techniques developed in this new deep learning paradigm up to 2018.



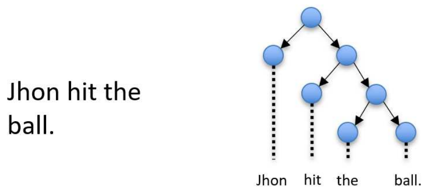
(a) Molecules as graphs



(b) n-body problem as graph



(c) Image as graph



(d) Text as graph

Figure 3.1: Examples of graphs. Adaptation from Figure 2 of [15].

Applications of Graph Neural Networks have been explored in a wide range of domains across supervised, semi-supervised, unsupervised and reinforcement learning settings. Due to their flexibility, they can obtain state-of-the-art performance in comparison with deep learning architectures in regression and classification problems [157]. In addition, their greatest potential lies in the processing of graphs, performing tasks such as node and graphs classifications and edge predictions. For example, predicting nodes or edges in a knowledge graph [190]. Next are listed some recent usages in different areas: learning the dynamics of physical systems (Battaglia et al., 2016 [14]; Sanchez-Gonzalez et al., 2018 [156]), predicting the chemical properties of molecules (Duvenaud et al., 2015 [48]; Gilmer et al., 2017 [55]) or even in imagery for classifying and segmenting images and videos (Wang et al., 2018 [189]; Hu et al., 2017 [73]). The works cited above are just a few examples of a much longer list. For the interested reader, Jie Zhou et al. in [204] review most of the relevant applications divided into three categories: structural scenarios, non-structural scenarios and other scenarios such as generative models and combinatorial optimization problems. Despite the daunting amount of applications, the current PhD will focus on the fields related to sensorised environments understood as explained in Chapter 1.

The remainder of this chapter will provide an intuitive explanation of how these models work. Following that, the GNN variants explored in the contributions of this thesis will be detailed. In the concluding subsection, some of the most relevant advantages and limitations of GNNs will be discussed.

3.1 An Intuition of how GNNs Work

It's been mentioned that a GNN can process graphs, but how is a graph defined in the context of a GNN? The ***graphs*** fed into a GNN can be formally defined as a two-element tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $v_i \in \mathcal{V}$ and edges $(v_i, v_j) \in \mathcal{E}$. Each node v_i is associated with a set of features in the form of a vector $\vec{h}_i$. In that sense, nodes can represent abstract information or specific entities in an environment, as we will see in the practical applications of the remaining chapters. It is up to the graph designer how to create these representations. Typically, the feature vectors are stacked vertically forming a feature matrix H of dimensions $N \times h_n$, where N is the number of nodes in the graph and h_n is the length of the feature vectors. On the other hand, edges link pairs of nodes, allowing the information interchange between their feature vectors. In some variants (see Section 3.2), the edges can also have attached a feature vector $\vec{e}$.

Mathematically, graphs can be expressed in different formats such as a list of nodes pairs representing the edges (sparse representation) as most programming frameworks do[1]. This format has the advantage of saving space when storing graph structure. However, for the sake of clarity, an adjacency matrix (dense representation) will be utilized to define the structure of the graph. The ***adjacency matrix*** A is NxN where N is the number of nodes in the graph. The position i, j of the matrix will store a 1 if there is a connection between the node i and j and 0 otherwise.

[1]For example, DGL framework for python uses a list of two-element tensors to store the graph: `https://docs.dgl.ai/en/0.4.x/generated/dgl.DGLGraph.edges.html`

$$A_{ij} = A_{ji} = \begin{cases} 1 & i \leftrightarrow j \\ 0 & otherwise \end{cases} \tag{3.1}$$

With the graph description established, we will discuss the process of information extraction by a GNN layer from these graph representations. The basic idea behind GNNs is to learn a representation of the nodes in a graph by aggregating information from the neighbouring nodes (and potentially edges). This is typically done by passing messages between nodes, where each message is computed as a learnable function of the current node's features and the features of its neighbouring nodes and edges. The messages are then combined to update the representation of the current node. After that, the representation is updated with an activation function. This process is repeated in the multiple layers of the network, and the final representations of the nodes are used for various tasks, such as node classification, link prediction, and graph classification. It is important to note that the graph structure of a graph neural network (GNN) remains consistent from layer to layer, with the only variation occurring in the features of the nodes. Additionally, it is worth highlighting that the input graph in a GNN can have an arbitrary number of nodes and edges, as the weights of the network are shared locally, as will be seen in more detail in the next section. This property provides an important advantage for a wide range of applications.

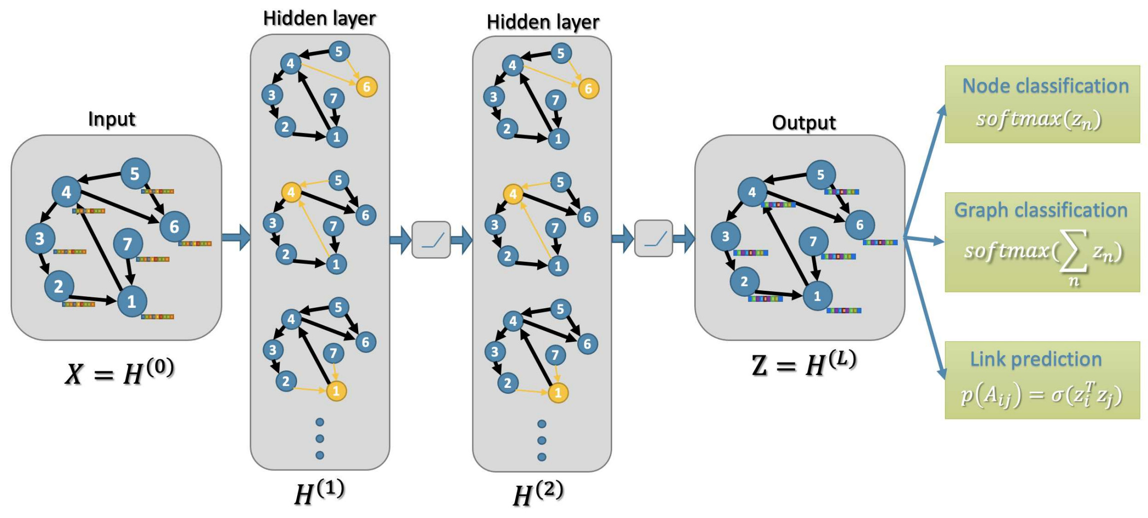


Figure 3.2: Pipeline of a graph neural network.

Figure 3.2 represents the pipeline of a GNN and how the final representation (output layer) can be used for the different tasks. For node classification, a classifier (for example a softmax layer or an MLP) is applied to the node features. On the other hand, graph classification is done by aggregating the features of the nodes first (for example summing or averaging them) and then applying the classifier. If we want to predict the probability of an edge connecting two nodes, it is possible to train an MLP yielding this probability and taking as input the features of the two nodes. Finally, it is worth noting that it is possible to perform a regression from a graph by configuring the layers of the network to have a single feature in the output representation of the nodes.

3.2 GNN Variants Explored in this Thesis

Due to the rapid blooming of Graph Neural Network models, it is out of the scope of this work to study all of them. This section shows the most relevant GNN variants for the present thesis, all of them producing cutting-edge results in the areas mentioned at the beginning of this chapter. This section provides more detail about how the information from the different nodes is aggregated in the GNN layers, starting with -in my opinion, the most basic architecture- the Graph Convolutional Network (GCN). For coding all these networks, the Python library *PyTorch*[2], which is well-known in the deep learning community, has been employed. Two main packages build on top of PyTorch for working with graphs and relational data: *Deep Graph Library (DGL)*[3] and *PyTorch Geometric*[4]. These libraries have many different tools to work with graphs as well as the implementation of the following GNN variants. Most of the applications of this thesis are developed with DGL but there are some examples developed with PyTorch Geometric for comparison purposes.

The increasing interest in GNNs has led to the development of numerous variants from various scientific domains, each addressing graph analysis from different perspectives. This has resulted in complex classifications with multiple subcategories, as shown in [204]. Nowadays, most of the non-dynamic (not leveraging the time evolution in the data) variants can fall into three different categories: convolutional, attentional and message-passing.

- **Convolutional:** Aggregate neighbours with averaged weights. They work well for homophilous graphs where edges encode label similarity. Very scalable and lightweight.

[2]Pytorch documentation: `https://pytorch.org/`
[3]DGL documentation: `https://www.dgl.ai/`
[4]PyTorch Geometric Documentation: `https://pytorch-geometric.readthedocs.io`

- **Attentional:** The aggregation is done with implicitly learned weights via an attention mechanism that is based on the nodes' features. They are also lightweight but at the same time, they achieve more expressiveness in the solutions giving more importance to some connections over others.

- **Message-passing:** The aggregation is done by sending across a message in the form of a vector from the source node to the receiver node. This message is computed by a learnable function. They perform better in complex problems. However, storing and computing an entire vector for each node of the graph can lead to scalability or learnability issues.

In this thesis, at least one model from each of these categories has been explored. For convolutional layers, work has been done with Graph Convolutional Networks (GCNs) and Relational Graph Neural Networks (RGNN). Within the attentional and message-passing categories, the Graph Attention Networks (GAT) and Message-Passing Neural Networks (MPNN) have been respectively explored.

3.2.1 Convolutional Networks

√

Graph Convolutional Networks (GCN)[5]: GCNs are first described in [90] where Kipf et al. use semi-supervised learning on three different datasets. Each node of the net is processed in parallel and gets an average sum of the feature vectors of the nodes linked to it. Then, this averaged vector is multiplied by the weights matrix and an activation function will yield the final updated output. This procedure is similar to applying a fully connected network per node but taking into account the information of the linked nodes. The intuition of GCN comes from applying convolutions on graphs in a similar way to how a CNN does in an image-like structure. Equation 3.2 describes how the output feature vector $h_i^{(l+1)}$ of a node i on layer $l+1$ is computed.

$$\vec{h}_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \frac{1}{C_{ij}} W^{(l)} \vec{h}_j^{(l)}\right) \tag{3.2}$$

[5]DGL reference documentation: `https://docs.dgl.ai/generated/dgl.nn.pytorch.conv.GraphConv.html`

In equation 3.2, $\mathcal{N}(i)$ is the set of nodes j so that an edge (j, i) exists in the graph, $W^{(l)}$ is the trainable weight matrix for the layer l, $\sigma(\cdot)$ is the activation function and C_{ij} is a normalisation parameter. You will notice that the equation is very similar to the equation used in an MLP but without taking into account the bias term. That performs the update of the feature vector for just one node, if we want to compute the update of all nodes at once we can use matrixes multiplication (Equation 3.3).

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{3.3}$$

Here, $H^{(l)}$ refers to the matrix resulting from vertically stacking all $\vec{h}_i^{(l)}$ vectors, $\tilde{A}$ is an adjacency matrix with self-edges and W is a weight matrix. $\tilde{D}$ is the degree matrix of $\tilde{A}$ and the opperation $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ normalizes the adgencency matrix according to its degree. The multiplication of the normalised A with H has the effect of aggregating the information of neighbouring nodes performing a sum as represented in Equation 3.2.

√

Relational Graph Convolutional Network (RGCN)[6]: This network is introduced by Shlichkrull et al. in [158] for predicting relations between entities and discovering new entity classes in knowledge graphs. The rationale behind Relational Graph Convolutional Networks (RGCN) is to treat each different edge with a different linear transformation, depending on the labels of the edges. These labels can only be types and are not meant to contain additional metric information (unlike the MPNN edge features explained in Section 3.2.3). They build on top of GCNs, using a different learnable weight matrix for each label type. The propagation model for the feature vector of the node i is shown in equation 3.4.

$$h_i^{(l+1)} = \sigma\left(\sum_{r \in R}\sum_{j \in N^r(i))} \frac{1}{C_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right) \tag{3.4}$$

Where $\mathcal{N}^r(i)$ is the set of vertices with an outgoing edge towards vertex i under the relation r and $C_{i,r}$ is a "problem-specific" normalisation constant. $W_r^{(l)}$ and $W_0^{(l)}$ are the learnable matrices for r-labelled edges and self-edges, respectively. For highly multi-relational

[6]DGL reference documentation: `https://docs.dgl.ai/generated/dgl.nn.pytorch.conv.RelGraphConv.html`

data, the original paper suggests using basis regularization by decomposing W_r to avoid an enormous number of parameters.

$$W_r^{(l)} = \sum_{b=1}^{B} a_{rb}^{(l)} V_b^{(l)} \tag{3.5}$$

Where B is the number of bases, $V_b^{(l)}$ are linearly combined with coefficients $a_{rb}^{(l)}$.

3.2.2 Attentional Networks

√

Graph Attention Networks (GAT)[7]: Velickovic et al. introduce this model in [182], which is one of the most expressive and most popular variants at the present date thanks to its attention mechanism. This net is similar to GCN with the difference that a learnable attention factor α is applied to the links. The network will learn the weights of each node along with the attention parameter of each link giving more importance to some connections over others. Additionally, GATs introduce a multi-head attention mechanism. K independent attention parameters are used in the same layer by concatenating the activations of each attention mechanism. The multi-head mechanism has proved to stabilize the learning process of self-attention.

$$h_i^{(l+1)} = \|_{k=1}^{K} \sigma\Big(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{k} W^{k,(l)} h_j^{(l)}\Big) \tag{3.6}$$

In Equation 3.6, the feature vector of node i is updated from the neighbouring nodes weighted by a learnable attention parameter α. The results are then concatenated at the output. The parameter α is calculated as follows:

$$\begin{gathered} \alpha_{ij} = softmax(e_ij) = \frac{exp(e_{ij})}{\sum_{k\in\mathcal{N}(i)} exp(e_{ik})} \\ where\ \ e_{ij} = \sigma\Big(\vec{a}^T [W\vec{h}_i \| W\vec{h}_j]\Big) \end{gathered} \tag{3.7}$$

e_{ij} indicates the importance of node j's features to node i.

[7]DGL reference documentation: `https://docs.dgl.ai/generated/dgl.nn.pytorch.conv.GATConv.html`

Although it has not been used in this thesis (largely due to its recent introduction), it is worth mentioning the creation of the second version of this model called GATv2 by Brody et al. in [23] in 2022. Brody's work shows that the attention performed by the first GAT version is limited since the ranking of the attention scores is unconditioned on the query node. They solve this problem by changing the order of operations in GATv2 to calculate e_{ij}:

$$e_{ij} = \vec{a}^T \sigma\Big(W \cdot [\vec{h}_i \| \vec{h}_j]\Big) \tag{3.8}$$

In this case, the parameters $\vec{a}$ are applied after the nonlinearity and the W parameters after the concatenation of the nodes' features.

3.2.3 Message-Passing Networks

√

Message-Passing Neural Network (MPNN)[8]: This variant is proposed by Gilmer et al. in [55] for molecule prediction. One of the most attractive characteristics of this variant is that it incorporates the possibility of adding features for the edges of the graph. Thus, the information of neighbouring nodes is aggregated taking into account the connection features. The architecture of this variant can be adapted to describe other different models of the literature. Specifically, in the contributions of this thesis, the adaptation for convolutions without hidden states for the edges has been used. This implies that the model does not learn the feature of the edges, but instead leverages this information from the input graph. In other words, the edge features are the same for the input and output graphs. Using the same equations from [55] for graph convolutions, first, the network calculates the messages from the surrounding nodes (Equation 3.9). These messages are computed from a learnable function $f_m(\cdot)$ (for example an MLP) taking as input the concatenation of the source node features $\vec{h}_i^{(l)}$, the destination node features $\vec{h}_j^{(l)}$ and the edge features $\vec{e}_{ij}$.

$$\vec{m}_i^{(l)} = \sum_{j \in \mathcal{N}(i)} f_m\left(\vec{h}_i^{(l)}, \vec{h}_j^{(l)}, \vec{e}_{ij}\right) \tag{3.9}$$

[8]DGL reference documentation: `https://docs.dgl.ai/generated/dgl.nn.pytorch.conv.NNConv.html`

After the messages of each edge are computed, the features of each node are updated with another learnable function $f_u(\cdot)$ as Equation 3.10 describes.

$$\vec{h}_i^{(l+1)} = f_u(\vec{h}_i^{(l)}, \vec{m}_i^{(l)}) \tag{3.10}$$

Note that the summation in Equation 3.9 acts as an aggregation function that can be replaced by any other aggregation operation depending on the necessities, such as average or maximum value.

3.2.4 Comparison Among Cited Models

It is interesting to include a comparison made in [118] of the performance of different GNNs in the task of generating a discomfort score indicating the disturbance to humans by a robot in a room. (This model is later used in the work developed in Chapter 4 to create a disruption maps dataset.)

Framework	Network architecture	Training Loss (MSE)
DGL	GCN	0.02283
DGL	**GAT**	**0.01701**
DGL	GAT2	0.01740
PyG	GCN	0.29778
PyG	GAT	0.01804
PyG	RGCN	0.02827
PyG	RGCN||GAT 1	0.02238
PyG	RGCN||GAT 2	0.02147
PyG	RGCN||GAT 3	0.0182

Table 3.1: Comparison table among different GNN structures. From [118]

As you can see in table Table 3.1, GAT achieves the better loss for this problem. However, this is not always the case, for example, RGCN yields better performance in the experiments conducted in Chapter 4. The most likely reason for the better performance of GAT in [118] is the rich information encoded in the features of the nodes and the reduced number of nodes. This makes it easier to infer the different kinds of relations without the need of adding information to the edges. On the other hand, the graphs used in chapter 4 have a significantly larger amount of nodes and the features in the gird nodes provide scarce semantic information to the network. Note that in Table 3.1, GAT2 refers to a network with two MLP layers to compute the messages instead of a single perceptron and not to the second version of Graph Attention Networks described earlier.

3.3 Advantages of GNNs

In light of the focus of this thesis, it is crucial to address the question of why GNNs are useful in many applications of the current deep learning panorama. This section aims to introduce the primary advantages of this new paradigm compared to traditional approaches for extracting information from graphs.

One of the primary advantages of GNNs is their ability to effectively process graphs. This is significant because there are a plethora of problems that can be naturally represented as graphs. As discussed at the beginning of this chapter (see Figure 3.1), molecules in chemistry, free body particles, maps and a large etc. can be represented as graphs. Images and sequences of text can be also easily translated into a graph, consequently, CNNs and recurrent neural networks can be formalized as a specific case of GNNs. Even transformers [180] are a specific version of GATs [85]. Therefore, advances in GNNs will translate into advances in other areas of deep learning.

Due to the inherent characteristics of graphs, classic ANNs cannot work with them efficiently. Three of these characteristics that GNNs are equipped to handle will be highlighted. The first one is that the size and shape of a graph can differ from another from the same distribution. A key advantage of GNNs is the flexibility that they offer in terms of input dimension. In contrast, traditional machine learning models such as MLPs or CNNs are trained with a fixed dimension of input variables. If the same trained architecture needs to be applied to inputs of different dimensions, the entire network must be retrained. GNNs, however, do not have this limitation and can handle input graphs with variable sizes. This allows for more efficient and versatile modelling of real-world problems.

The concept of isomorphic graphs is an important characteristic in graph theory, implying that two visually distinct graphs can be structurally identical following the application of an isomorphism function. To illustrate, even if the nodes of the graphs are repositioned, the structure remains unchanged despite the nodes being in a different order. As the nodes of a graph are not presupposed to follow any specific order, Graph Neural Networks (GNNs) are, by necessity, required to exhibit permutation equivariance. This means that any permutation matrix P applied to the node features H or the adjacency matrix A must not affect the result of the network:

$$f(PH, PAP^{-1}) = f(H, A) \tag{3.11}$$

The third and arguably the most challenging characteristic is that the graph structure is non-euclidian. CNNs can operate over images because the underlying grid of the image information is fixed and allows Euclidian operations. Concepts such as the Euclidian distance between two graphs or two nodes are not defined. GNNs can perform convolutions in a non-euclidian space accounting for complex higher-order information such as motifs or substructure counts that standard CNNs cannot discover.

This thesis explores the application of GNNs to sensorised environments, where graphs seem to be convenient representations of the problems. These problems deal with different input sizes, for example, the application in Chapter 6 can use a different number of sensors (cameras) while the applications in Chapters 4, 5 and 7 have to account for a variable number of entities in their environments. Furthermore, the environments in Chapters 4 and 5 encompass underlying interactions without a fixed pattern, which can only be represented with non-euclidean structures such as graphs. In these chapters, it's demonstrated that GNNs are capable of overcoming these pitfalls and achieving state-of-the-art results for the specific applications.

3.4 Limitations of GNNs

Despite the proven effectiveness of GNNs for a multitude of graph-related tasks, they still possess substantial limitations. This section goes through the most relevant drawbacks of learning on graphs with GNNs and explains how they can affect the results of the thesis.

3.4.1 Problems with the Depth, Over-Smoothing and Over-Squashing

One of the most relevant impediments is the difficulty of training deep GNN networks [133]. CNNs experienced a meteoric rise in popularity with the increasing number of layers leading to the development of very deep networks. For instance, popular networks such as VGG16 and VGG19 [162] using 16 and 19 layers respectively, can extract richer information from images than their shallower predecessors. The visualization of the features learned by CNNs from face images indicates a progressive increase in complexity, starting from simple geometric shapes (edges and lines) and culminating in complete facial structures. However, this compositionality does not appear to be achievable with graphs. As noted in [38], it appears that GNNs cannot count induced subgraphs for any connected pattern of 3 or more nodes.

The main two problems concerning excessive depth in GNNs are the over-smoothing of the layers' output [133] and the over-squashing of the feature vectors [7]. Regarding the over-smoothing of the output, the node features become smoother when using more layers and the performance decay as they converge to a similar vector losing expressiveness. The over-squashing of the feature vectors is a bottleneck problem, which results in the compression of information from an exponential number of neighbours into fixed-size vectors. This problem limits the propagation of information between distant nodes in the graph even if the GNN has many layers. These challenges are an important consideration when designing GNN architectures and must be addressed to improve the performance of the model.

CNNs overcame depth-related problems such as vanishing gradients and overfitting (due to a large number of parameters) thanks to the implementation of residual connections and regularization techniques. Significant effort has been done to apply similar techniques to graphs allowing the training of deeper GNNs. For example, several works applying residual connections to GNN layers [109][31][199], graph normalization techniques such as NodeNorm [205] or PairNorm [203] and regularization techniques in works like DropEdge [150] or DropNode [76]. Chen et al. [32] gather most of the literature on these tricks comparing their performance in a fair benchmark. While these techniques allow training deeper GNNs, they fail to demonstrate significant gains.

As depicted in Table 3.2, sourced from [32], the accuracy of all models and datasets

Backbone	Settings	Cora			Citeseer			PubMed			OGBN-ArXiv		
		2	16	32	2	16	32	2	16	32	2	16	32
GCN	Residual	74.73	20.05	19.57	66.83	20.77	20.90	75.27	38.84	38.74	70.19	69.34	65.09
	Initial	79.00	**78.61**	**78.74**	70.15	**68.41**	**68.36**	77.92	**77.52**	**78.18**	70.16	70.50	70.23
	Jumping	80.98	76.04	75.57	69.33	58.38	55.03	77.83	75.62	75.36	**70.24**	**71.83**	**71.87**
	Dense	77.86	69.61	67.26	66.18	49.33	41.48	72.53	69.91	62.99	70.08	71.29	70.94
	None	**82.38**	21.49	21.22	**71.46**	19.59	20.29	**79.76**	39.14	38.77	69.46	67.96	45.48
SGN [193]	Residual	**81.77**	82.55	80.14	71.68	71.31	71.00	78.87	79.86	79.07	69.09	66.52	61.83
	Initial	81.40	**83.66**	83.77	71.60	**72.16**	**72.25**	**79.11**	79.73	79.74	68.93	69.24	69.15
	Jumping	77.75	83.42	**83.88**	69.96	71.89	71.88	77.42	**79.99**	**80.07**	68.76	70.61	70.65
	Dense	77.31	81.24	77.66	70.99	67.75	66.35	77.12	72.77	74.84	**69.39**	**71.42**	**71.52**
	None	79.31	75.98	68.45	**72.31**	71.03	61.92	78.06	69.18	66.61	61.98	41.58	34.22

Table 3.2: Table from [32] showing the accuracy results (%) of different skip connection mechanismos among different datasets (Cora, Citeseer, PubMed, and OGBN-ArXiv) and different network sizes (2/16/32 layers of GCN and SGC).

generally decreases as the number of layers increases. Although this table illustrates the results obtained using various random dropping techniques, other tables in [32] also display the same declining trend in performance with deeper networks. Chen and colleagues aim to identify the optimal combination of strategies to train deep Graph Neural Networks (GNNs) in [32]. Despite their efforts to enhance the efficacy of GNNs with deeper architectures, their findings do not demonstrate substantial improvements. Consequently, whether deeper GNNs will yield superior results remains an open research question.

As discussed thus far, the performance of Graph Neural Networks (GNNs) tends to deteriorate as the networks become deeper. Nonetheless, the question remains: do we actually require deep GNNs? The answer to this inquiry largely depends on the characteristics of the graphs present in the dataset. Deeper networks perform more aggregations, which enables nodes that are further away in the graph to share information. However, if all nodes in the graph are within a few hops of each other, or communication between distant nodes is not necessary, deeper networks become irrelevant. For instance, in social networks, predictions often rely solely on short-range information from a node's local neighbourhood and do not benefit much from additional remote information. In such cases, shallower and wider networks (i.e., networks with more neurons per layer) may prove to be the best solution. To validate the expressive capability of the networks employed to solve the problems addressed in this thesis, a modest experiment is conducted, which is detailed in Appendix B. The results reveal that for the problems tackled in this work, shallow networks possess sufficient power to produce state-of-the-art results.

3.4.2 Problems with Isomorphism and Different Types of Aggregators

Another substantial limitation appears in the task of graph classification with GNNs. It happens when trying to distinguish between topologically similar graphs or validate if two graphs are isomorphic. Two isomorphic graphs have the same connectivity and differ only by a permutation of their nodes. The task of verifying if two graphs are isomorphic is called the graph isomorphism test and it is not clear what is the complexity of this problem. The classical Weisfeiler-Lehman (WL) isomorphism test [191] in 1968 claimed to solve the problem in polynomial time. However, it has been seen that the WL test fails in some simple cases as the one represented in Figure 3.3.

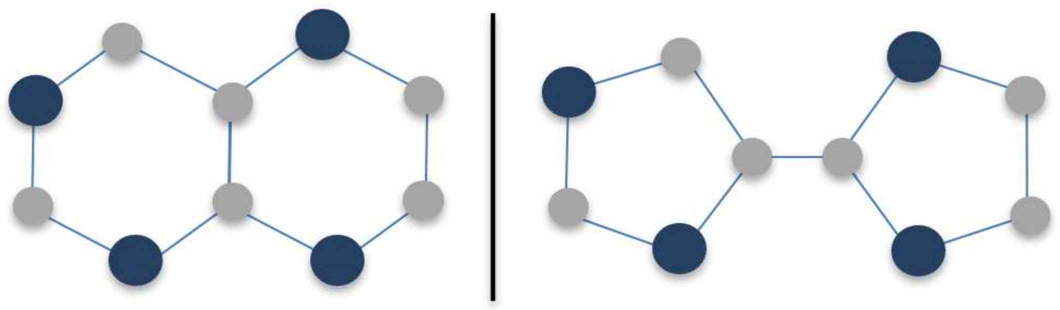


Figure 3.3: Non-isomorphic graphs that fail at the WL isomorphism test. They are both classified as the same type of graph.

Similar to GNNs, the WL test iteratively updates a given node's feature vector by aggregating feature vectors or hashes of its network neighbours. What makes the WL test so powerful is its injective aggregation update that maps different node neighbourhoods to different feature vectors. For example, using an injective aggregation function as the sum, the MPNN is at most as powerful as the WL tets and some popular choices for aggregators used in the literature such as maximum or mean are actually strictly less expressive than WL [198]. The work done in [198] introduces a new network named Graph Isomorphism Network (GIN) that uses summation as the aggregation function for countable features. They show that its discriminative/representational power is equal to the power of the WL test.

More powerful versions of the WL test, such as the k-WL test [75], recolour k-tuples of vertices of a graph at each step according to some neighbourhood aggregation rules and stops upon reaching a stable colouring. If the histograms of colours of the two graphs are not the same, the graphs are deemed not isomorphic; otherwise, the graphs are possibly (but not necessarily) isomorphic. The greater the k the more expressive the test, thus, there exist examples of graphs on which k-WL fails and $(k+1)$-WL succeeds, but not vice versa. There are works on GNNs applying the k-hop in the k-WL test that are strictly more powerful than message-passing architectures. One such first architecture, k-GNN, was proposed by Morris et al. [125]. The problem with these GNN variants is their resource consumption making them impractical for real applications.

The work of Corso et al. in [42] extends the study in [198] to the continuous space with more aggregators. They prove the following theorem:

Theorem 1 *(Number of aggregators needed). In order to discriminate between multisets of size n whose underlying set is R, at least n aggregators are needed.*

I have deemed it important to mention this problem of GNNs since it is one of the main limitations in the literature. However, there is not any application in this thesis for graph classification. All of the work developed in this thesis do regressions on graphs or edge prediction using a continuous representation of the features. For this kind of application, the graph isomorphism problem is not as relevant. Besides, in practice, higher capacity does not necessarily offer better generalisation (sometimes quite the opposite).

3.5 Conclusions

This chapter highlights the vast potential of Graph Neural Networks, but the PhD focuses specifically on their application in sensorised environments, where their advantages (discussed in Section 3.3) can be harnessed to great effect. The upcoming chapters will delve into the use of powerful GNN variants (outlined in Section 3.2) to solve problems in sensorised environments, to achieve state-of-the-art results.

While there are still limitations in this field, as discussed in Section 3.4, this thesis hopes to contribute to the wider adoption of these remarkable ANNs, hopefully, overcoming these limitations in the prompt future.

Chapter 4

Human-Aware navigation with Static Cost Maps

The work presented in this chapter has been adapted from the following publications:

[146] **Rodriguez-Criado, Daniel**, P. Bachiller, and L. J. Manso. Generation of human-aware navigation maps using graph neural networks. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 19–32. Springer, Cham, 2021.

This chapter investigates the generation of cost maps that represent areas of disruption to humans in indoor environments using Graph Neural Networks (GNNs), focusing on static scenarios. Chapter 5 extends this research by introducing improvements in dataset generation and the model architecture to accommodate dynamic environments. As examined further in this text, cost maps are essential for facilitating Human-Aware Navigation (HAN), which requires robots to not only identify humans as dynamic entities but also consider their interactions with other individuals and objects in the surrounding environment.

Companion and assistive robots are continually integrating as part of our society as we become increasingly reliant on these technologies [64, 34, 21]. These robots need to follow social conventions to avoid disturbing humans, to behave predictably and to increase their acceptability [99]. This requires robots to be aware of their surroundings, the people nearby and their activities. In a natural environment, our paths are obstructed by other humans, obstacles and structural elements such as walls and pillars. We can navigate through these obstacles seamlessly however, it is a core consideration when developing the algorithms for robots. They must rely on robust navigation models that allow them to move safely among humans. Even if the robot does not collide or harm any person, it is essential that humans feel safe and comfortable next to it to facilitate its acceptance in our society. Additionally, another core consideration of these models is efficiency. If we are to continually integrate robots into our daily life, they need to be of benefit to humans and one aspect of this is the ability to achieve the desired goal in the least time possible. Ultimately, all of the aforementioned considerations are taken into account in the problem of Human-Aware Navigation (HAN). Comprehensive surveys on the extensive research conducted in this field can be found in [99, 144, 123, 28].

Incorporating efficiency and social compliance into HAN models can prove challenging, primarily due to the multitude of variables involved. These factors encompass not only the environment and the initial and final positions of the robot but also the humans present in the scene, their activities, and preferences. Consider a situation wherein the robot's goal lies on the opposite side of a narrow corridor, with a human situated in the middle. The robot should be able to account for the individual's personal space, corridor width, and walking speed, among other factors, when planning its path. Despite substantial efforts (see Section 4.1), it is no trivial task to combine all these variables within an analytic algorithm that produces an optimal path for the robot. Manual design is resource-intensive, difficult to debug, and susceptible to overlooking crucial variables. Furthermore, such solutions often disregard interactions or make simplistic assumptions that machine learning can overcome by considering more complex factors. Converting these social variables into hard-coded constraints leads to intricate situations, frequently resulting in what is known as the *Freezing Robot Problem (FRP)* [175]. In this scenario, the planner determines that all forward paths are unsafe, causing the robot to freeze in place (or execute short, unnecessary manoeuvres) to avoid collisions.

Recent research directions address HAN using Deep Learning (DL) based models, which can be trained to produce near-optimal actions for the robot during the navigation process.

A DL approach circumvents the need for creating complex analytic models to solve the problem. For instance, several methods within the Deep Reinforcement Learning (DRL) paradigm have been applied to HAN. The majority of these DRL contributions offer feedback to the agent through a reward function [37], which is typically handcrafted and based on the distance between the robot and people, disregarding any other social variables. Such simplistic functions often result in suboptimal robot paths. Moreover, end-to-end RL approaches that do not explicitly search at run time lack a clear cost function that can be visualised and analysed. This makes them less transparent and more difficult to integrate with other algorithms, imposes hard-coded constraints (e.g., arbitrary boundaries) and reduces their explainability. Explainability is a crucial aspect for intelligent autonomous agents to interact with humans effectively [103].

A specific branch of DRL, known as Inverse Reinforcement Learning (IRL) [130], aims to teach the robot to emulate expert trajectories by learning the reward function. Although a significant number of these approaches utilise human trajectory patterns as expert trajectories, arguably, the robot should not necessarily move in the same manner as a person. To illustrate this, consider a scenario in which the robot is large, heavy, and potentially dangerous to people in the event of a collision. In this case, it would be undesirable for the robot to approach a human as closely as another person might.

An alternative solution using IRL could involve generating trajectory examples by having a person manually control the robot, typically in a simulation. However, this approach presents certain limitations. Firstly, IRL algorithms assume that expert trajectories are generated by an optimal policy. It is a strong assumption that humans can produce examples of optimal policy without any errors. Furthermore, for most behaviours, there are numerous suitable reward functions. The set of solutions often contains many degenerate solutions, such as assigning zero rewards to all states. Although IRL constitutes a promising area of research for HAN, it still exhibits certain limitations that require refinement, as highlighted in this paragraph.

Another potential solution for robot navigation involves the generation of disruption maps, which can serve as cost maps for planners such as A^*[131] or RRT[104] in subsequent stages to determine the most optimal path. Cost maps offer broader information to the planner in a more efficient manner compared to querying a model to get the discomfort score for each potential future step of the robot. Commonly employed approaches, such as Gaussian Mixture Models (GMM) [181], attempt to predict areas of disruption by modelling with Gaussian kernels the positions of humans. However, GMMs cannot directly incorporate additional variables beyond the position coordinates of the entities and are unable to approximate any function with merely one kernel per person. This highlights the limitations of handcrafted algorithms when managing multiple variables. Bearing all these factors in mind, this chapter posits that a learning model, specifically a GNN, could encode many of the previously mentioned social variables into cost maps, thereby assigning the navigation algorithm the task of avoiding disruption areas. The primary challenge lies in harnessing an appropriate dataset from which the learning algorithm can more effectively learn to model discomfort compared to simple queries.

As discomfort is a subjective human sensation, one possible approach to generating a dataset involves directly inquiring individuals about their feelings in the vicinity of a real robot. The human brain implicitly accounts for all environmental factors (or the most relevant) when generating a sense of unease. Although asking participants about their feelings in the presence of a real robot manoeuvring within a room would be ideal, this is not feasible in an actual study for several reasons. Firstly, it is inadvisable to recreate collision cases (negative samples are beneficial for dataset training purposes) as this would pose a danger to participants. Secondly, training a deep learning model necessitates the collection of a substantial amount of data, which is a costly and time-consuming endeavour when employing a real scenario procedure. As an alternative, experiments can be conducted in simulated scenarios, which are then labelled by real individuals from an external perspective. This approach was employed in the *SocNav1* dataset [119], which was used to train a model in [118] to predict discomfort scores, as described in Section 4.2.

The model outlined in [118] is capable of generating a cost map by querying every potential position of the robot within a room. However, the inefficiency of this process renders it unsuitable for real-world applications. The main objective of this chapter is to propose a novel architecture for modelling robot disruption in human comfortability that can efficiently generate two-dimensional cost maps for HAN taking into account interactions, which is an under-researched topic in the literature. Previous studies (see section 4.1) have attempted to address interactions, but they either rely on arbitrary decisions or suffer from significant efficiency limitations when applied to path planning.

This chapter presents two main **contributions** to the field of human-aware navigation: **a)** a novel method for bootstrapping two-dimensional datasets from point-wise datasets that capture the social context of human-robot interactions; and **b) SNGNN-2D** (Social Navigation Graph Neural Network with 2-dimensional output), a hybrid architecture that integrates GNNs and CNNs to generate two-dimensional cost maps based on the data available to the robot. These cost maps can be used as a cost function for planning human-aware navigation trajectories since they can be generated on-the-fly. SNGNN-2D can take as input a graph encoding the positions of the entities in the room and semantic information such as the kind of interactions between these entities to directly output the 2D cost map.

The underlying assumption is that *SNGNN-2D* has the potential to generalize over various scenarios the discomfort scores in a 2D image format, suitable for navigation purposes. The initial hypothesis poses that these newly derived cost maps can significantly enhance the performance of navigation algorithms HAN metrics, surpassing the capabilities of existing state-of-the-art models. The experiments conducted in section 4.4 demonstrate the accuracy, efficiency and performance of the proposed model compared to a baseline Gaussian Mixture Model-based (GMM) algorithm. The software for generating the two-dimensional dataset and implementing *SNGNN-2D* is publicly available in an open-source repository, along with all the data required to replicate the experiments[1].

This chapter employed a base model that generates a two-dimensional dataset based on static information from simulation frames [118]. However, this approach can also be applied to dynamic scenarios with a different model that produces a score for each situation, as discussed in Chapter 5. In such cases, the graph input for the GNN contains not only static information but also dynamic information (e.g., relative motion of humans and robots).

4.1 Related Work

One of the main challenges in robot navigation is to account for the social aspects of human-robot interaction. Previous research has mainly focused on collision avoidance with

[1]Source code of the project: `https://github.com/gnns4hri/sngnn2d`

objects, humans and walls, treating humans as dynamic obstacles. However, Human-Aware Navigation (HAN) goes beyond this by incorporating the social norms and expectations that govern human behaviour and spatial relations. Two major approaches have been developed in this direction: proxemics and social force models (SFM). Proxemics draws on psychological and sociological studies [66, 144] to define three criteria for socially acceptable robot navigation: comfort, naturalness and sociability. Comfort involves respecting personal and interpersonal spaces, avoiding collisions and not obstructing peoples' paths. Naturalness aims to emulate human motion and behaviour to reduce confusion and increase efficiency [99]. Sociability entails following social conventions such as not interrupting people who are engaged in conversation. SFM, on the other hand, was introduced by [69] and extended by [176] and [137]. SFM and its variants [51] model humans as repulsive forces that induce robots to keep a distance from them. These models have been used to simulate pedestrian dynamics and enable robots to imitate them. However, both proxemics and SFM have limitations as they rely on hand-crafted rules and parameters. As previously argued, this makes them prone to errors and inconsistencies due to the high number of variables involved in social situations [178].

This chapter and other comparable works follow the approach of creating models using DL or a combination of handcrafted models and DL to design navigation systems. Many of these works employ CNNs to infer environmental features from raw images of the scene. A case of this approach is Perez-Higueras et al. [139], who merge a fully convolutional network (FCN) that learns from expert demonstrations with an RRT* planner. Their work, nonetheless, has some shortcomings. The dataset is scarce because it is difficult to obtain human-generated trajectories. Additionally, it does not consider social cues such as human-human and human-object interactions, which are vital for human-aware navigation [123].

Deep Reinforcement Learning (DRL) has been effectively applied to navigation in crowds among learning-based models. Chen et al. [37] present a policy that enables a robot to navigate through an environment with many pedestrians while following social norms. In this case, social conventions are encoded using a reward function based on geometric features. Therefore, some of the limitations of hand-crafted methods discussed earlier can also affect this approach. It is worth noting the challenge of designing reward functions that promote social behaviour.

Inverse Reinforcement Learning (IRL) [130] enables agents to learn from human experiences. Unlike reinforcement learning, IRL infers reward functions from human demonstration. A recent example is Sun et al. [170], who use a combination of A* and IRL for social navigation using only the robot's sensors as input. IRL has also been employed to generate social cost maps for robot navigation in [179]. However, IRL-based models implicitly learn the relations and interactions between humans and human-object, which can result in poor performance in dense crowds. To address this issue, [36] and [29] propose the combination of GNNs and reinforcement learning for crowd navigation. They present an end-to-end approach where the model directly produces the robot control signals. Thanks to the use of GNNs, the dynamics of crowds are better represented and captured. Nevertheless, the reward function is hand-crafted and only considers the distance to the nearest human. Moreover, these works do not account for explicit interactions among people.

RL or IRL end-to-end approach allows for directly producing the final signals controlling the robot. Any information about the environment that may affect the control of the robot has to be initially considered as there is no possibility to include that information at a later stage in end-to-end approaches. This means that not only information about the people and their relations with the environment is important, but also other factors such as the size and shape of the objects. An alternative is to build a cost map that can be used by a planner to generate a minimum cost path. Cost maps can combine information from different sources and are more easily explainable than final control signals which, may be helpful during the development of the system.

The model presented in this chapter, *SNGNN-2D*, is a DL-based approach that generates cost maps for human-aware navigation. It overcomes some of the aforementioned limitations related to the ability to incorporate symbolic information that cannot be naturally represented using a single vector or grid, such as interactions. Specifically, *SNGNN-2D* combines GNNs and CNNs to generate a cost map from a graph representing the different elements of a room as well as the relationships between them. Unlike pure CNN models, which are not designed to capture the relational information among the elements in the scene [15], *SNGNN-2D* can integrate both geometrical and relational data, which are crucial in HAN.

Moreover, *SNGNN-2D* is not dependent on specific features of the environment (e.g., the textures of the floor and walls) that would prevent obtaining a general model that could be applicable in other scenarios. This advantage is demonstrated empirically in Chapter 5, where the second version of *SNGNN-2D* results are compared with those obtained by a representative CNN-based model and show that the CNN model can not extrapolate to the mentioned changes in the scenario. The combination of GNN and CNN layers provides a general solution to the generation of maps for social navigation that integrates both geometrical and relational data. *SNGNN-2D* is trained using a map-like 2D dataset bootstrapped from a single-point model developed in a previous work [119]. In such a model (*SNGNN*), for any given graph describing a scenario, the network generates a single scalar estimation of how disruptive the robot is overall for the people in the scenario. To generate a cost map from these single scores, it is necessary to query the model for each of the cells in the cost map. The time required for generating the cost map this way makes this solution unsuitable for real-time applications. *SNGNN-2D* uses a generated dataset that contains scenario-maps tuples which are computed by sampling the output of *SNGNN* for every robot position (see Section 4.3.1).

4.1.1 Graph Neural Networks Applied to Human-Aware Navigation

A number of recent machine learning-based approaches leveraging structured data for social navigation have been recently published. A GNN model integrated with a Deep Reinforcement Learning (DRL) algorithm based on Monte Carlo Tree Search was presented in [29]. It utilises a graph-based model to detect the implicit relations between the humans in a room. Interactions are useful to predict future human trajectories. For instance, interacting pedestrians generally behave differently than those who do not interact. This phenomenon is also exploited in [35], where a GCN-based DRL leverages the gaze of humans to estimate interactions and predict their trajectories. These works consider human-robot and human-human relations but disregard interactions with objects or obstacles that could be exploited. Moreover, the DRL algorithms in [29] and [35] use a simple handcrafted reward function based on the minimum distance between the robot and the humans that disregards any other social information such as how densely populated the room is (distance restrictions are usually eased in crowded spaces). Due to the variety of different scenarios and factors to consider, handcrafting a reward function that complies with social rules seems prohibitively complex and time-consuming.

A model combining a CNN and a GNN to learn an action policy for multi-robot navigation is presented in [110]. The CNN extracts features from local observations of the environment, and the action policy for the robot swarm is computed from those features using GNN. Although safety and collision avoidance are taken into account, the approach only considers humans as obstacles.

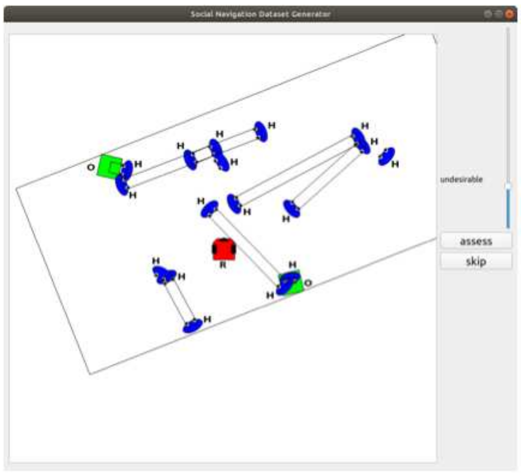
Other works use GNNs for reasoning and perception in a domain related to social navigation. A significant amount of them directs their focus to the prediction of pedestrians' paths [183], [77] or [65]. A particularly noteworthy contribution by Agrawal et al. [5] uses knowledge graphs and machine learning to estimate the location of objects in a room. The use of GNNs for these tasks allows extracting additional information of the environment and from the crowd such as relations between people and between humans and objects. However, none of the previous works tackles the problem of modelling discomfort.

4.2 Model to Predict Discomfort Scores

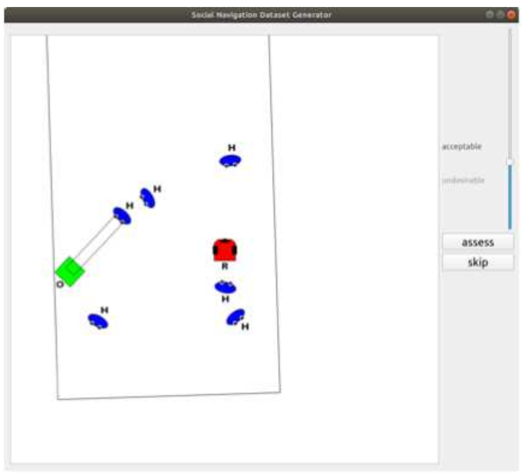
This section summarises the work conducted in [118], which presents a GNN model, termed *SNGNN*, designed to predict a general discomfort score for individuals in a room containing a robot and static objects. The input of this model is a graph containing metric and semantic information of the scenario. While not a direct contribution to this thesis, this work lays the foundation for a series of studies on HAN developed herein.

SNGNN [118] was trained using the *SocNav1* dataset [119]. This dataset comprises single-frame scenarios depicting a room populated with humans, objects, and a robot, as illustrated in Figure 4.1. Blue ellipses represent humans, green rectangles objects, and the red square denotes the robot. Two types of interactions are represented by black lines: human-human interactions (see Figure 4.1a) and human-object interactions (see Figure 4.1b).

Three participants involved in the creation of the dataset assigned to each scenario a score between 0 and 100. A score close to 100 suggests that the robot is not causing any inconvenience to the individuals in the room, while a score close to 0 indicates an unwelcome



(a) Human-humans interaction.



(b) Human-object interactions.

Figure 4.1: Scenario representation from the SocNav1 dataset [119]. The 2D images depict blue ellipses for humans, green squares for objects, and a red square for the robot. Human-human and human-object interactions are represented by two parallel lines between the entities. The slider on the right is used to provide the discomfort score caused by the robot in the scenario.

situation or a collision with a human. The slider next to the scenario in Figure 4.1 was used to determine the score when acquiring the dataset. To ensure the consistency of the three users labeling, both inter-rater and intra-rater agreement were quantitatively assessed using the linearly weighted kappa coefficient [41].

The creation of *SocNav1* was motivated by two key factors. Firstly, considering the current technology readiness level, the expected behavior from robots may differ from what is anticipated from interactions among humans. This divergence highlights the necessity to understand and align robot behavior with human expectations. Secondly, *SocNav1* aims to evaluate the capacity of robots to gauge the discomfort their presence might evoke among humans. This ability holds significance for robot navigation systems as it aids in estimating path costs based on human comfort levels.

Prior to training the model, the dataset information must be converted into a graph that the GNN can process. In *SNGNN*, graphs resemble the one shown in Figure 4.2. Each entity in the room is represented by a node in the graph, including wall segments and a room node. The robot serves as a global node, connecting to other nodes and gathering global information. Interactions between two entities are represented by linking their respective nodes with an edge in the graph. Node features encompass a one-hot encoding vector that identifies the type of entity it represents, along with metric features such as the entity's coordinates within the room.

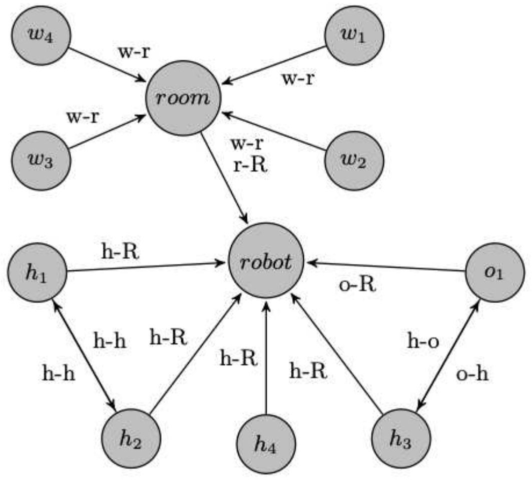


Figure 4.2: Graphs employed in [118] to encode the information in each scenario.

Moreover, beyond effectively generalizing over the discomfort scores provided in *SocNav1*, SNGNN also considers the interactions between entities. For example, if the robot is positioned between two individuals engaged in conversation, the model's predicted score appropriately reflects this scenario by yielding a lower score, acknowledging the presence of the ongoing interaction. This nuanced understanding of social dynamics represents a bias successfully extracted by the model from the data labeled by subjects in *SocNav1*.

However, the model does possess some notable limitations. One such limitation arises from the training data, which only offers single-frame scenarios with static elements, while in real life, velocities are crucial for the network to estimate future entity positions. Another constraint is the absence of semantic information in the data. It would be beneficial to have details such as the type of interaction (e.g., talking, shaking hands, walking together), the scenario context (e.g., office, hospital, hallway), and the timing of collisions between entities, to name a few examples. This information could be conveniently incorporated into either edge features (for interaction type) or node features (for scenario type). These two limitations will be addressed through the development of a new dataset, as introduced in Chapter 5.

A further significant limitation, addressed in this chapter, is the limited utility of a single score per scenario for navigation algorithms. As previously mentioned, *SNGNN* can be employed to produce two-dimensional cost maps by querying the network for every potential position of the robot within the room. This method generates discomfort scores, assigning a value to each pixel in the cost map (refer to Section 4.3.1). However, the primary issue with creating maps in this manner is the time it consumes. Given that a network query is required for each pixel in the map, the entire procedure takes an average of 140 seconds in the setup utilised in [118], thus preventing the system from being used in real time.

This time constraint served to motivate the subsequent work presented in this chapter. The following text outlines the process of utilising *SNGNN* to bootstrap a new dataset containing 2D images, which will be employed for training a model capable of generating 2D discomfort maps. As previously stated, these cost maps can be used by a planner to provide safe paths for the robot whilst adhering to the social conventions within the room.

4.3 Generation of Disruption Maps from Static Data

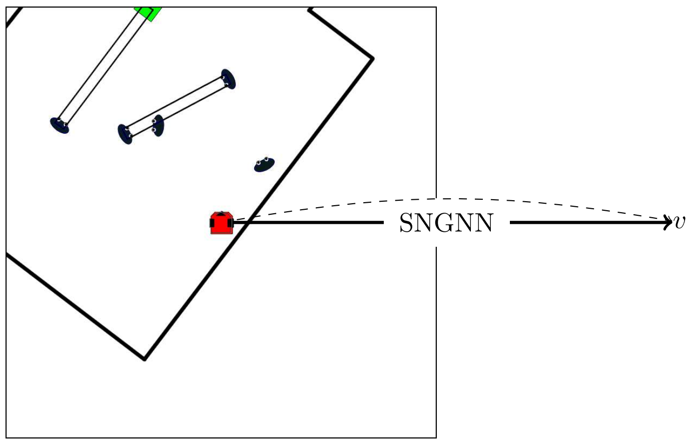
This section presents the main technological contributions of the chapter proposal: a) the technique employed to generate the dataset; b) the scenario-to-graph transformation; and c) the architecture of the proposed SNGNN-2D model.

4.3.1 2D Dataset Generation

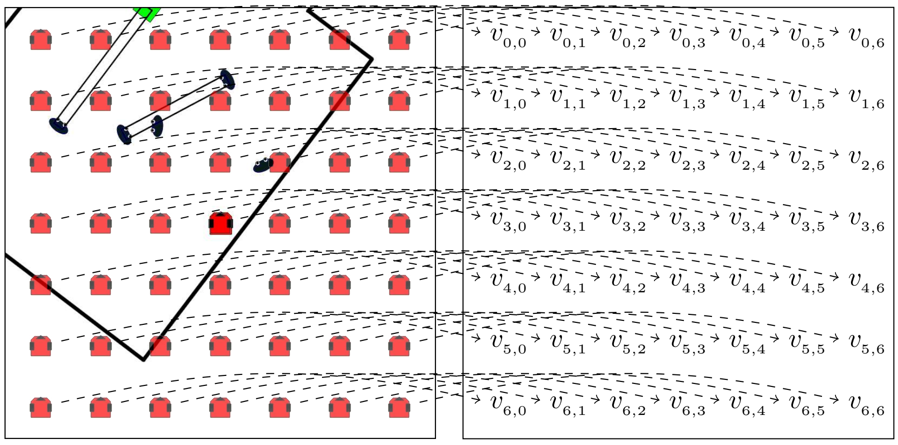
Acquiring two-dimensional cost or disruption maps to create datasets for learning introduces several challenges. One major issue is that requesting individuals to provide a cost map rather than a single score per scenario would be more time-consuming. Additionally, the accuracy of the responses will depend on the subjects' ability to graphically represent their preferences, which may vary. Furthermore, ensuring the subjects' inclination and motivation to remain engaged in the task is another challenge that needs to be considered.

From a DL perspective, when considering approximately equal time commitments and efforts for generating responses, providing a single scalar for each scenario would result in responses for a greater number of scenarios. This, in turn, would arguably increase the variability in the input scenarios and make the model less susceptible to overfitting.

A dataset that includes scalars as output data cannot be directly used to train a model that provides a two-dimensional output. Therefore, in this case, we utilized a model that



(a) SNGNN can be used to estimate the disruption caused by the robot given a particular scenario. In the scenario on the left, humans are shown as ellipses, objects as squares and interactions as parallel lines. The value V represents the response of SNGNN to the presence of the robot.


$v_{0,0}$ $v_{0,1}$ $v_{0,2}$ $v_{0,3}$ $v_{0,4}$ $v_{0,5}$ $v_{0,6}$
$v_{1,0}$ $v_{1,1}$ $v_{1,2}$ $v_{1,3}$ $v_{1,4}$ $v_{1,5}$ $v_{1,6}$
$v_{2,0}$ $v_{2,1}$ $v_{2,2}$ $v_{2,3}$ $v_{2,4}$ $v_{2,5}$ $v_{2,6}$
$v_{3,0}$ $v_{3,1}$ $v_{3,2}$ $v_{3,3}$ $v_{3,4}$ $v_{3,5}$ $v_{3,6}$
$v_{4,0}$ $v_{4,1}$ $v_{4,2}$ $v_{4,3}$ $v_{4,4}$ $v_{4,5}$ $v_{4,6}$
$v_{5,0}$ $v_{5,1}$ $v_{5,2}$ $v_{5,3}$ $v_{5,4}$ $v_{5,5}$ $v_{5,6}$
$v_{6,0}$ $v_{6,1}$ $v_{6,2}$ $v_{6,3}$ $v_{6,4}$ $v_{6,5}$ $v_{6,6}$

(b) The expected 2D outputs are generated by performing multiple queries to SNGNN, shifting the scenario around the robot.

Figure 4.3: The process used to bootstrap the two-dimensional dataset: a) how a single SNGNN query works; b) how to generate two-dimensional outputs.

provides single value estimations (*SNGNN* [118]) and sampled its output by shifting the robot's position to bootstrap a two-dimensional dataset. The process, as depicted in Figure 4.3, is as follows: given a scenario and a particular position of the robot, *SNGNN* estimates a score of the disruption caused by the robot to the humans in the scenario. To generate a map from this model, multiple queries are made to *SNGNN* for each scenario, moving the robot to each potential position. This process generates a two-dimensional output that represents a cost map per scenario. Ultimately, the dataset comprises pairs of data: graphs containing scenario information (outlined in the subsequent section) serving as inputs for training the model, and the corresponding anticipated outputs, i.e., the bootstrapped 2D images representing the cost maps.

For each scenario in the bootstrapped dataset, a matrix of 73×73 samples is generated. A total of $37,131$ scenarios were randomly generated following the same strategy as *SocNav1* [119]. The dataset split for training, development, and testing includes $31,191$, $2,970$, and $2,970$ scenarios, respectively. Given the relatively high number of samples and the randomness of the selection process, each set in this split can be considered sufficiently representative of the entire set.

4.3.2 Scenario to Graph Translation

In order to use GNNs, it is essential to convert the input data, which is not presented in the form of a graph, into a graph-like structure. In the approach presented here, the process follows similar steps as in [118] (see Section4.2), with the exception that an additional grid of 18×18 nodes is added, and its values are passed to the CNN layers of the architecture and decoded into the final output. This grid of nodes and the architecture designed in this chapter stand as a significant technological contribution of the thesis, enabling the efficient transformation of graphs into images.

The first part of the graph, which coincides with [118], is the scenario graph that represents the entities in the room and their relations using a node per entity, such as room, humans, walls, and objects. The walls are divided into segments, creating a node for each of these segments. A global *room* node connects to every other node in the graph to facilitate the use of global information in the room using fewer GNN layers. This is a crucial aspect to consider as GNNs tend to suffer from poor performance with a large number of layers, as discussed in Chapter 3. The human-to-human and human-to-object interactions, if they exist, are represented as edges among the respective nodes. The scenario graph can be seen in the top half of Figure 4.4.

The mentioned grid is a lattice of interconnected nodes, structured to represent the area of the room surrounding the robot by associating them to 2D coordinates. The connection of the grid nodes is represented in Figure 4.4. The number of nodes of this grid and the area they cover are hyperparameters that can be tuned to reach a trade-off between performance, computation time, and area coverage. The x, y coordinates of a grid node in row i and column j from the robot's perspective are computed as in equation 4.1, where N is the

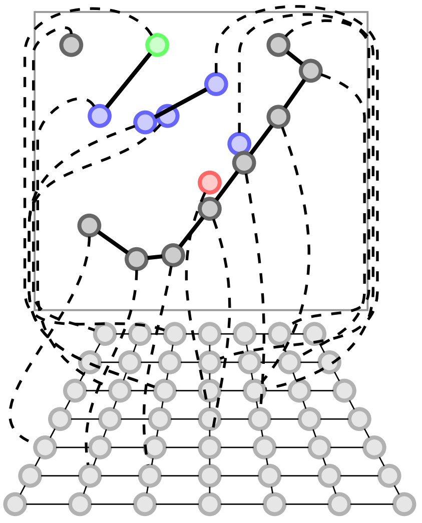


Figure 4.4: Graphical representation of the final graph for the scenario depicted in Fig. 4.3a. The nodes for the room, humans, objects and walls are drawn within the square at the top of the figure. The node for the room is shown in red (in the centre), blue nodes represent humans, green nodes represent objects and wall nodes are drawn in dark grey. Grid nodes (in lighter grey) are connected among them and form a squared mesh area. The nodes in the scenario graph connect to the closest node of the grid (see dashed arrows). All nodes in the graph are bidirectionally connected to the room node but are not drawn to facilitate the visualisation. Edge types are not displayed to avoid clutter.

width and height of the lattice and W is the side of the squared area covered by the grid.

$$\begin{aligned} x &= \frac{W\,(\lfloor (N-1)/2 \rfloor - i)}{N-1} \\ y &= \frac{W\,(j - \lfloor (N-1)/2 \rfloor)}{N-1} \end{aligned} \tag{4.1}$$

In the final graph, the scenario graph and the grid are combined using additional edges to connect the entities from the scenario graph to the closest nodes in the grid. This results in a single graph that is used as input to the model. Fig. 4.4 provides a representation of what the final graph for the scenario depicted in Fig. 4.3a looks like.

Each node in the graph is assigned a feature vector with 21 dimensions. The feature vector $h_i^{(0)}$ for the i-th node is created by concatenating a one-hot encoding and type-specific metric information, as shown in equation 4.2. Here, t_i represents a one-hot encoding to distinguish between the five types of nodes: human, object, room, wall, and grid.

$$h_i^{(0)} = (t_i \parallel p_i \parallel o_i \parallel r_i \parallel w_i \parallel g_i) \tag{4.2}$$

The sub-vectors p_i, o_i, and w_i store the metric information for human, object, and wall nodes, respectively. These sub-vectors are four-dimensional and include the 2D coordinates of the entity and the cosine and sine of its orientation for normalisation purposes. Additionally, there are sub-vectors r_i and g_i that contain metric properties for room and grid nodes, respectively. In particular, the room feature vectors store the number of humans in the room, whereas the grid vector stores the 2D coordinates of the corresponding node. In cases where the metric vectors do not correspond to the type of node, the sub-vectors are filled with zeros. Although it is possible to merge the metric features of different entity types into a single metric vector, we chose to keep them separated to help the neural network combine the information as required to obtain the final output. It's essential to clarify that the assumption within this context is that the information required to construct the graphs is known. In real scenarios, other models, such as the one developed in Chapter 6, are necessary for detecting people's positions and their interactions before employing *SNGNN-2D*.

As seen in the next section, the GNN layer model that yielded the most promising results in this work is RGNN, which allows for the utilisation of labelled edges as discussed in Section 3.2. This entails that each edge is associated with a specific edge type that corresponds to the nodes it links and their respective direction. For example, a connection between a *human* node and an *object* node differs from that of an *object*-to-*human* connection. This labelling scheme also extends to the edges connecting the scenario graph and the grid graph. However, the edges in the grid are labelled according to the direction of the connection to account for their relative positions, namely, up, down, left, and right.

4.3.3 Architecture Description

The architecture that yielded the best results after hyperparameter tuning is illustrated in Fig.4.5, and it consists of three segments. The first segment comprises a sequence of 8 Relational Graph Convolutional Network layers (RGCN), with the number and size of these layers determined via hyperparameter tuning (as described in section 4.4.1). The output of this segment is a graph that preserves the same nodes and structure as the input graph, but with feature vectors that have been reduced from 21 to 7 dimensions. In the second segment, nodes that are not part of the grid are removed to connect the output of the last

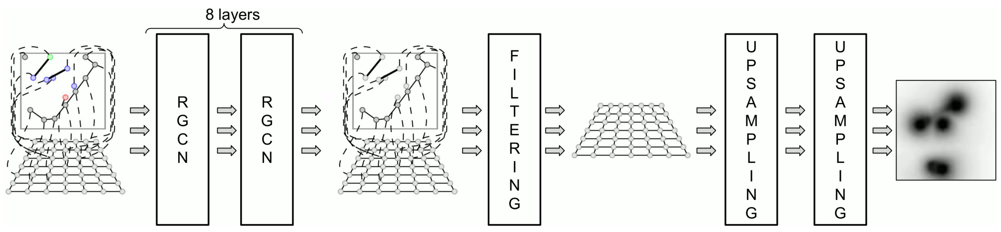


Figure 4.5: Architecture of the SNGNN-2D model.

GNN layer to the input of the first CNN layer. Specifically, a grid comprising 18×18 nodes is employed in the model, which generates an $18 \times 18 \times 7$ tensor that is utilized as input for the third and final segment of the architecture. This third segment consists of a sequence of two transposed convolutional layers that generate the final 73×73 output that is observed in the bootstrapped dataset. The combination of RGCN and CNN layers, linked through a node filter layer that generates cost maps, is the second contribution of this chapter.

4.4 Experiments

To validate the proposal, this section includes experimental results regarding the accuracy of the proposed model, its time performance and a comparison to other approaches.

4.4.1 Training Results

The model was trained as a regression model using MSE (Mean Square Error) as the loss function where the expected output is the bootstrapped cost maps. After running 150 training tasks to optimise the hyperparameters of the architecture using random search, the best model achieved an MSE of **0.00071**, **0.00112** and **0.00114** for the training, development and test splits of the bootstrapped dataset, respectively. The model reached the best performance on the development dataset after 35 epochs. The 8 R-GCN layers transform the input feature vectors from 21 into 7 dimensions. The 2 transposed CNN layers of the model with the lowest test MSE have kernel sizes of 5 and 3, with a stride of 2 and a padding of 1. The non-linearity of the best-performing model was ELU. Figure 4.6 depicts three scenarios and the corresponding output of the *SNGNN* (sampled) and *SNGNN-2D* models.

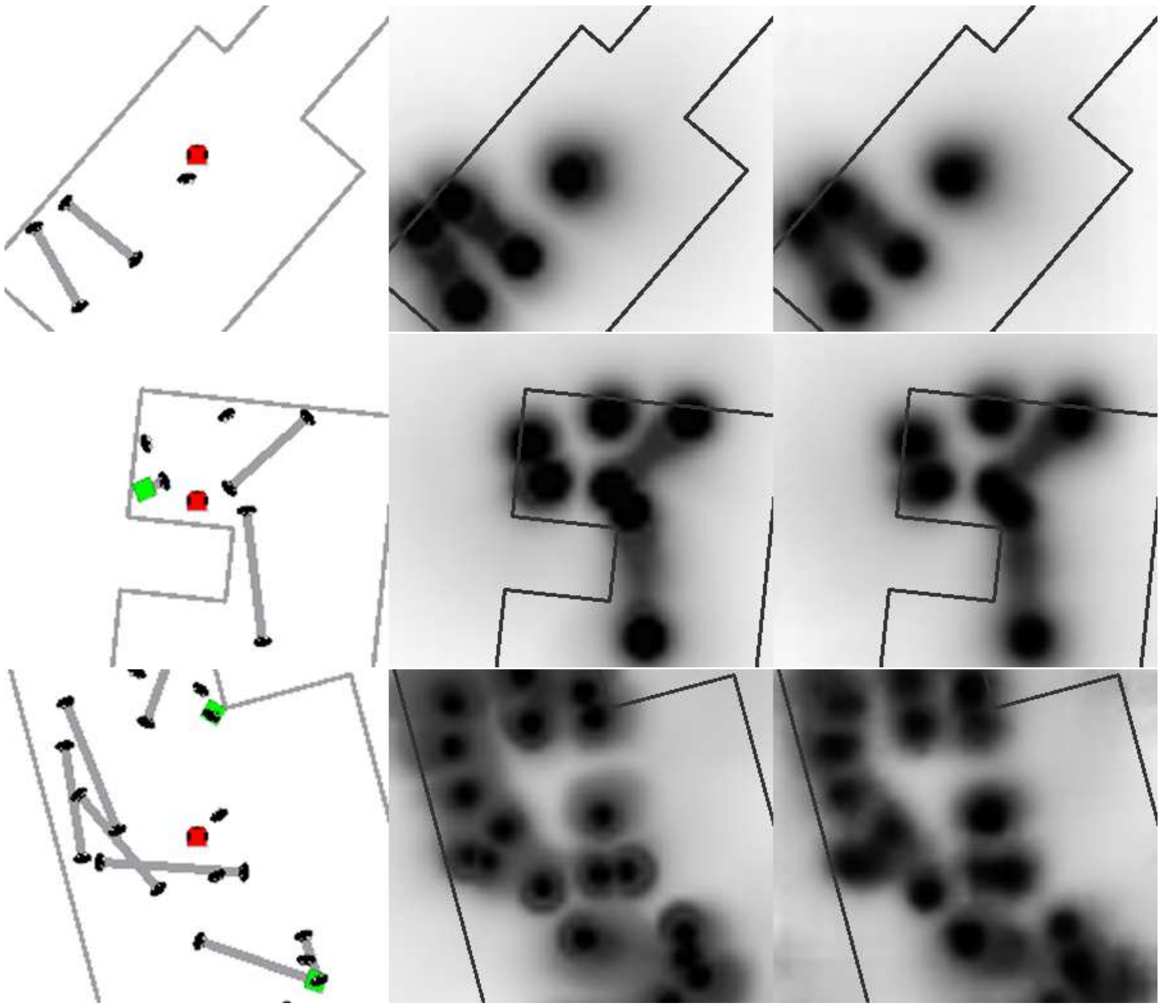


Figure 4.6: Results obtained for 3 different scenarios. On the left, are representations of the rooms. The central images correspond to the bootstrapped labels used as ground truth. On the right, is the output generated by SNGNN-2D. Ideally, the second and third columns should be equal.

4.4.2 Comparison against the Reference Dataset (SocNav1)

Given that the bootstrapped dataset does not contain information gathered directly from users but from the output of a zero-dimensional GNN model (one single value as output), comparing *SNGNN-2D* against the test data of the bootstrapped dataset could lead to unrealistic results. To provide a realistic evaluation of our model with the previous zero-dimensional model, the original *SocNav1* test dataset was used.

Considering that *SNGNN-2D* delivers a complete image while SocNav1 offers a singular score, this section computes the MSE using only the central pixel of the image., which corresponds to the robot's position. Based on user-assessed data only, the MSE of *SNGNN-*

2D computed for the *SocNav1* test set was 0.01873. The *SNGNN* version used to generate the bootstrapped dataset performed slightly better than the one reported in the original paper, with an MSE on the *SocNav1* test set of 0.0198. *SNGNN-2D* not only achieves better time efficiency but also achieves slightly better accuracy on the test set when compared to *SNGNN*. This accuracy is in turn noticeably superior to the one obtained by the non-ML based approach in [181] (0.12965 as reported in [118]), which is used as a reference method in section 4.4.4 to evaluate navigation results. This indicates that the cost maps generated using *SNGNN-2D* reflect more precisely how people would feel in different situations.

4.4.3 Real-Time Evaluation

The time required by SNGNN to generate a 73×73 frame in a 6th generation Intel i7 computer with a Geforce GTX 950M is 37.55 seconds (it requires 5329 SNGNN queries). The time required by SNGNN-2D to generate a similar output is **0.045** seconds, with just one query (three orders of magnitude difference). The substantially decreased generation time of *SNGNN-2D* enables its application in real-time scenarios, such as the experiment detailed in the following section.

4.4.4 Comparison against GMM-Based Methods

This section presents a comparative evaluation of the effectiveness of SNGNN-2D, through simulated navigation results, with the human-aware navigation approach proposed in [181] that relies on Gaussian Mixture Models (GMMs).

The experiments were conducted under simulated environments using the SONATA [11] toolkit. This toolkit is built on top of PyRep [83] designed to simulate human-populated navigation scenarios and to generate datasets. SONATA provides an API to generate random scenarios including humans, objects, interactions, the robot and its goals. The walls delimiting a room are also randomly generated considering rectangular and L-shaped rooms. SONATA also provides real-time access to information on the elements in the environment and their properties. This information is utilised by the tested methods to create a cost map that is integrated into a control system responsible for planning the robot's minimum cost path using $A*$ and guiding the robot towards its goal. In Chapter 5, more information about this tool will be provided as it is used to generate the training dataset.

The simulation environment provides all the required information for creating the graphs in the navigation scenario. However, in a real-world setting, the robot would operate in a sensorised environment where data from various sensors (such as RGBD cameras, microphones, and lasers) would be preprocessed by other models to obtain the necessary global knowledge for graph generation, including the position of entities, relations, and global information. For instance, a model like the 3D pose estimator described in Chapter 6 could be utilised to obtain human positions in a room as demonstrated in the experimentation section of Chapter 5.

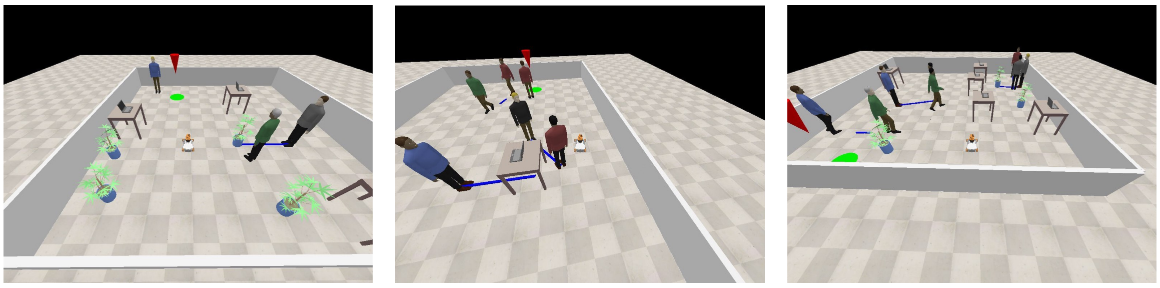

(a) Example scenario A **(b)** Example scenario B **(c)** Example scenario C

Figure 4.7: Examples for each of the three scenarios used for the experiments. Scenario A (Fig. 4.7a) has 2 standing humans and 1 walking human, scenario B (Fig. 4.7b) with 4 standing humans and 2 walking humans and scenario C (Fig. 4.7a) with 5 standing humans and 3 walking humans. The blue lines on the floor represent interactions and the green circle is the robot's goal.

According to the number of humans in the room, three different types of scenarios were tested: rooms with 2 standing humans and 1 walking human (S_A), rooms with 4 standing humans and 2 walking humans (S_B) and rooms with 5 standing humans and 3 walking humans (S_C). All the scenarios included a randomly generated number of objects, room shape and wall length. The number of interactions between humans or humans and objects was also randomly generated. Figure 4.7 depicts example frames for each of the three scenarios. For each type of scenario, each method was executed in 50 different simulations to cover a wide range of situations. The results of applying each method were evaluated using identical metrics as outlined in [181] to ensure a comprehensive comparative analysis:

- $\boldsymbol{\tau}$: navigation time
- $\boldsymbol{d_t}$: travelled distance
- $\boldsymbol{CHC}$: cumulative heading changes

- $\boldsymbol{d_{min}}$: minimum distance to a human
- $\boldsymbol{si_i}$: number of intrusions into the intimate space of humans (closer than **0.45**m)
- $\boldsymbol{si_p}$: number of intrusions into the personal space of humans (closer than **1.2**m)
- $\boldsymbol{si_r}$: number of intrusions into an interaction (closer than **0.5**m)

Table 4.1 shows the mean and the standard deviation of these metrics using the GMM-based method and *SNGNN-2D*, considering separately each group of scenarios. The best value of every metric for each type of scenario has been highlighted in bold. For the first two types of scenarios (S_A and S_B) the mean values of most of the metrics can be considered comparable, although *SNGNN-2D* produces better results according to the travelled distance (d_t) and the cumulative heading changes (CHC). More variability is observed in the GMM-based approach as can be seen by the standard deviation of each parameter. In addition, for complex scenarios (S_C) greater differences can be observed between the two methods, showing that the proposed model behaves in a more socially acceptable way in crowded environments. Another important result is that the time required to generate a cost map using GMM substantially increases with the number of people and relations, whereas the impact of the complexity of the scenario in the proposed model is negligible.

	GMM			SNGNN-2D		
	S_A	S_B	S_C	S_A	S_B	S_C
$\tau(s)$	12.16±7.0	**13.12** ± 6.7	18.9±10.2	**12.11** ± 4.7	13.75±5.2	**15.99** ± 6.2
$d_t(m)$	4.92 ± 2.5	4.88 ± 2.8	5.7 ± 3.1	$\mathbf{4.14} \pm 1.2$	$\mathbf{4.53} \pm 2.2$	$\mathbf{4.67} \pm 1.9$
$CHC(rads)$	4.36 ± 2.9	4.44 ± 2.1	5.99 ± 3.3	$\mathbf{2.87} \pm 1.4$	$\mathbf{3.18} \pm 1.3$	$\mathbf{3.64} \pm 1.6$
$d_{min}(m)$	$\mathbf{1.53} \pm 0.7$	$\mathbf{1.24} \pm 0.5$	0.98 ± 0.4	1.48 ± 0.7	1.12 ± 0.5	$\mathbf{1.01} \pm 0.4$
$si_i(\%)$	0 ± 0	$\mathbf{0} \pm 0$	0.2 ± 0.91	0 ± 0	0.11±0.73	$\mathbf{0} \pm 0$
$si_p(\%)$	12.0±27.2	**18.1** ± 27.2	**30.2** ± 32.4	**11.2** ± 16.5	22.6±25.8	30.3±25.4
$si_r(\%)$	0.37 ± 1.6	$\mathbf{0.18} \pm 0.9$	1.66 ± 5.9	$\mathbf{0} \pm 0$	0.39 ± 1.7	$\mathbf{0.55} \pm 2.4$

Table 4.1: Mean and Standard Deviation ($M \pm SD$) of the navigation metrics for each group of scenarios using GMM and SNGNN-2D.

4.5 Conclusions

As shown in section 4.4, the combination of arbitrarily structured and grid nodes in a two-staged architecture integrating GNN and CNN layers achieved competitive results considering both MSE and the properties measured in the navigation simulations. The reduction in time from an average of 37.55 seconds to 0.045 seconds in comparison to *SNGNN* is also remarkable, allowing the model to be used in real-time applications.

SNGNN-2D has a significant limitation, which is the absence of dynamic properties such as movement in the input data. In Chapter 5, this limitation will be addressed by creating a dataset that takes into account the velocity of the robot and humans, which is not considered in the current dataset (*SocNav1*) used to bootstrap the 2D dataset.

In this chapter, navigation results comparing *SNGNN-2D* with a Gaussian Mixture Model are presented. The results show that *SNGNN-2D* outperforms the GMM, particularly when the number of people in the scenario increases. In Chapter 5, a comparison against a pure CNN model will be conducted, although this comparison may be considered unbalanced for several reasons. Firstly, the CNN is not able to encode explicit relations like the GNN. Additionally, the GNN is provided with explicit information extracted from other sources, such as the position of entities in the room, while the CNN would have to implicitly extract this information from the raw data. Finally, CNN's training would be scenario-specific since it depends on the position of the cameras and the textures of the environment. On the other hand, the GNN facilitates working with structured data and higher lever information not relying on visual information such as textures. This allows the model to be deployed in environments with similar sensors. All these limitations will be highlighted in the comparison experiment in Chapter 5.

It is worth noting that the method followed in this chapter can be used to generate other kinds of maps with completely unrelated applications by applying the process to other datasets or even realistic-looking images as demonstrated in Chapter 7.

Chapter 5

Generation of Dynamic Cost Maps for Human-Aware Navigation

Part of the work presented in this chapter has been adapted from the following publications:

[10] P. Bachiller, **Rodriguez-Criado, Daniel**, R. R. Jorvekar, P. Bustos, D. R. Faria, and L. J. Manso. A graph neural network to model disruption in human-aware robot navigation. *Multimedia Tools and Applications*, pages 1–19, 2021.

[11] R. Baghel, A. Kapoor, P. Bachiller, R. R. Jorvekar, **Rodriguez-Criado, Daniel**, and L. J. Manso. A toolkit to generate social navigation datasets. In *Workshop of Physical Agents*, pages 180–193. Springer, 2020.

Part of the work in this chapter is also planned for submission to the *Journal of Ambient Intelligence and Humanized Computing* under the title *"Generation of Dynamic Human-aware Navigation Maps using Graph Neural Networks".*

Despite the remarkable improvement in speed when using a GNN-CNN combination for cost map generation, the model in the previous chapter *SNGNN-2D* presents several limitations. On the one hand, some limitations stem from the dataset used for training. The scenarios in *SocNav1* [119] only provide static information lacking any data on velocities, which are highly informative for navigation and path prediction models. Moreover, the interactions between entities indicate connection but do not provide any semantic information (e.g., two humans talking, humans walking together, humans shaking hands). Semantic information enriches the input and therefore the features of the GNN, potentially yielding more accurate results. On the other hand, limitations exist in the model itself. *SNGNN* and *SNGNN-2D* can only process static data and therefore they provide a score for a single frame. Both the pipeline and the generation of graphs have to be modified to take into account the dynamic of the environment. Addressing this limitations is the main motivation of the work carried out in this chapter.

In the preceding chapter, we examined two of the three most salient advantages of GNNs in the HAN domain: Firstly, GNNs permit a flexible number of input features, which proves practical in scenarios where there is a variable number of entities, such as humans and objects. Secondly, as GNNs can accept a graph as input, they capitalise on the relationships between entities, which are represented as edges connecting nodes in the graph. This explicit consideration of interactions results in the model attaining a more comprehensive understanding of the human-human and human-object interactions within the room. In this chapter, among other contributions, we explore a third advantage, which is that the direct utilisation of structured data introduces an additional degree of abstraction. This supplementary abstraction layer renders the system scenario-agnostic, meaning that the model operates with pre-processed raw data, independent of the particularities of the scenario. Owing to the abstraction power of GNNs, better performance across different scenarios is expected (enhanced generalisation) compared to an end-to-end approach. It is worthwhile to conduct a comparison experiment to address this hypothesis.

This chapter aims to address the previously mentioned limitations through three **primary contributions**. The first contribution is the development of a novel dataset, the **SocNav2 dataset**, comprising brief videos of a 3D environment that integrates the velocities of entities within the room. In addition to the dynamic context, *SocNav2* offers several improvements over its predecessor, such as an extended scoring system that accounts for robot movements and objectives, as well as more realistic scenarios and interactions (refer to Section 5.2). The second contribution encloses the introduction of an innovative model, **SNGNNv2** (Section 5.4), for generating discomfort scores from dynamic scenarios, serving as the second version of the static model discussed in Chapter 4 (*SNGNN* [118]). *SNGNNv2* adopts a similar approach to its predecessor but with notable enhancements. It considers two distinct scores to assess different facets of social navigation, and the model is trained utilising dynamic scenes in which humans and the robot are in motion, addressing the primary limitation of *SNGNN*. For the last contribution *SNGNNv2* is employed similarly to its predecessor to bootstrap a new dataset of images, used for training a new model that generates disruption maps, referred to as **SNGNN-2Dv2** (Section 5.5), which constitutes the second version of *SNGNN-2D*. Both these new models rely on processing a graph constructed from information preprocessed by additional models using information from the sensors in the environment.

The primary motivation behind this new dataset and models is the hypothesis that incorporating environmental dynamics will significantly enhance performance in social metrics for Human-Aware Navigation (HAN), as demonstrated in the experiments of this chapter. Furthermore, it is worth noting that *SNGNNv2* and *SNGNN-2Dv2* have distinct primary objectives. On the one hand, *SNGNNv2* is designed to evaluate social or human-aware navigation systems involving robots, proposing an innovative quantitative metric derived from qualitative considerations not previously employed in the literature. On the other hand, *SNGNN-2Dv2* generates cost maps that can be directly utilised for HAN algorithms. While the goals of both models differ considerably, they are included within the same chapter because the 2D version builds upon the discomfort score predictor, and both models focus on dynamic scenarios as opposed to the model created in Chapter 4.

The **additional contributions** of this chapter encompass a new method of creating graphs to consider the dynamics of the environments (Section 5.3) and two new experiments. The first experiment (Section 5.5.4) compares *SNGNN-2Dv2* with a CNN-based method for the same task of generating cost maps. This comparison highlights some of the CNN limitations mentioned, for instance, the CNN-based model performs poorly in different rooms with varying floor and wall patterns, while the GNN remains unaffected (appearance invariant). The second experiment (Section 5.5.4) tests the cost map generation model with a real robot in an actual environment. The map is utilised by the robot's navigation planner to follow a safe path adhering to social conventions.

5.1 Related Work

This section expands upon the literature review in Section 4.1, focusing on approaches for addressing HAN in dynamic scenarios through the use of cost maps. Recent surveys on HAN highlight the advantages of employing maps for robotic navigation [123, 28], particularly in the context of SLAM (Simultaneous Localization and Mapping) systems, which are widely implemented in contemporary commercial robots.

The review presented in [28] categorises cost maps used in HAN into three types: metric, semantic, and social maps. The authors acknowledge that distinctions among these categories are often ambiguous within the literature. Nevertheless, a more pronounced distinction exists between metric maps and the other two types. Metric maps focus exclusively on the geometric aspects of the environment, whereas semantic and social maps introduce semantic information, providing an additional layer of abstraction. Besides the advantages discussed in earlier chapters, this extra layer of abstraction also facilitates the transfer from simulation to real-world scenarios [123]. The experiment in Section 5.5.4 demonstrates the performance of *SNGNN-2Dv2* (trained using simulated data) within a real-world environment, employing an actual robot.

Numerous studies have explored the generation of cost maps that take into account the velocities and dynamics of environments for human-aware navigation. For instance, works such as [45, 93] extend the application of Gaussian Mixture Models (GMMs) to model areas of disruption in dynamic environments, considering the velocities of pedestrians. However, algorithms that rely on handcrafted social constraints, such as these, encounter the limitations outlined in Chapter 4.

Laible et al. [101] put forward a method for semantic robot localisation using spatio-temporal classification. The process begins with spatial classification, wherein the input is partitioned into a grid, with each grid cell assigned a label such as asphalt, cobblestones, grass, or gravel. Following this, the method's temporal aspect utilises visual odometry to merge the derived maps. The labelled maps are subsequently projected onto the grid, and a probabilistic criterion refines the grid labels by considering neighbouring cells. It is important to note that these maps treat people as mere dynamic obstacles, without incorporating relational or other semantic information. Furthermore, occupancy grids are known to have inherent limitations, including resolution constraints, memory consumption, and computational complexity.

In [49], the authors first model personal space and group interaction as social costs based on pedestrian perception, subsequently generating multi-layer dynamic cost maps. These maps incorporate social costs at various timesteps, derived from pedestrian trajectory predictions, which provide social constraints for global path planning. The global path planner then searches for the optimal state using a heuristic cost function based on the multi-layer dynamic cost maps. Despite its merits, this work models disruption areas with analytic functions, which are more susceptible to errors and inconsistencies and a DL-based approach. Nevertheless, the concept of multi-layer cost maps proves valuable and has been implemented in the experiments of this chapter. In the real-world scenario demonstrated in Section 5.5.4, the final cost map employed by the robot is a superposition of the robot map (see Appendix A) and the discomfort map generated by *SNGNN2d-v2*.

Alternative approaches for generating cost maps that consider environmental dynamics utilise graph structures. Hemachandra et al. [70] propose a semantic framework that models the environment based on natural language descriptions and scene classifications. The topology of the resulting graph contains nodes representing the robot's trajectory, while the edges indicate connectivity between the nodes. The temporal component is involved in updating the graph topology by taking into account previous metric exteroceptive sensor data, scene appearance observations, and natural language descriptions. Similarly, [94] capitalises on the temporal component, introducing a time-evolving navigation graph that delivers a semantic topology of the explored area and the connectivity among detected places in terms of inter-place transition probability.

Both aforementioned works demonstrate the advantages of employing graphs for incorporating semantic information into the model and explicitly accounting for element interactions. However, it is important to note that these studies do not specifically address the HAN problem, as they do not consider humans as entities in their navigation models.

Finally, it is worth mentioning that cost maps can also be created from visual features using end-to-end deep learning models for image generation. Section 2.4.2 offers a comprehensive overview of the most prominent technologies, including VAEs, GANs, and diffusion/denoising models. As previously mentioned, the main limitations of these models are their adaptability to new scenarios and poor performance in modelling interactions between entities in the environment. Owing to GANs' ability to generate high-resolution images relatively quickly, a state-of-the-art GAN model is selected for comparison with *SNGNN-2Dv2*. Specifically, the model developed in [81], also known as *pix2pix*, is chosen for this purpose.

5.2 SocNav2 Dataset

SocNav1 (Section 4.2) was developed to learn and benchmark disruption estimation functions for social navigation. *SocNav2*, presented in this chapter, shares the same objective as its predecessor; however, it incorporates the velocity and trajectory of the robot and surrounding humans among other advantages. Like *SocNav1*, *SocNav2* contains scenarios with a robot in a room, alongside various objects and individuals who may potentially interact with other objects or people. In *SocNav2*, the room also includes a landmark that constitutes a goal position to be reached by the robot. Any existing human-human or human-object interactions are noted in the scenarios. However, unlike in *SocNav1*, these notes contain semantic information indicating the nature of the interaction and the interactions are not limited to entities facing each other. For instance, there is a type of interaction of two or more humans walking together.

SocNav2 provides 13406 scored samples of dynamic scene sequences. Each sample consists of 35 “snapshots” of a scene of a room with a moving robot, objects and potentially moving humans, taken during a time interval of 4 seconds. Each *SocNav2* sample includes scores for two social navigation-related statements: “*the robot does not cause any disturbance to the humans in the room*” (***Q1***) and “*the robot is moving towards the goal efficiently, not causing any disturbance to the humans in the room*” (***Q2***). The scores range from 0 to 100, considering the following reference values:

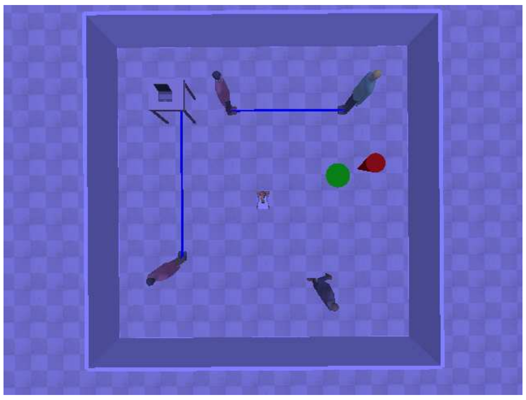
- 0: Unacceptable
- 20: Undesirable

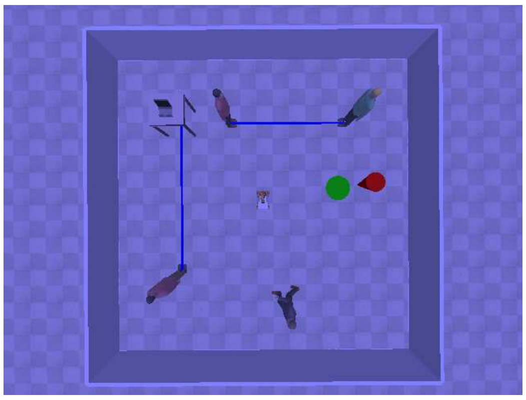
- 40: Acceptable
- 60: Good
- 80: Very good
- 100: Perfect

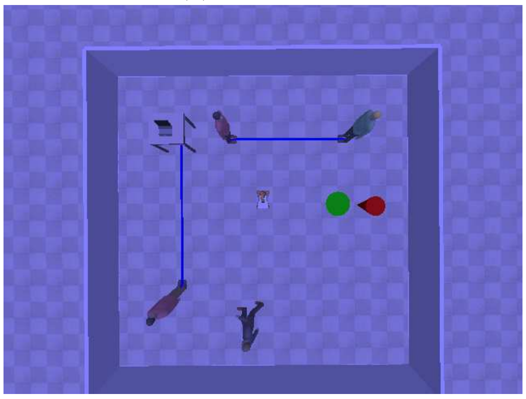
The scenarios compiled in *SocNav2* have been generated using *SONATA* [11]. *SONATA* is a toolkit built on top of *PyRep* [83] and *CoppeliaSim* [149], designed to simulate human-populated navigation scenarios and generate datasets. It offers an API to generate random scenarios, incorporating humans, objects, interactions, the robot, and its goals. The walls delineating a room are also randomly generated, considering both rectangular and L-shaped rooms. While *SONATA* exclusively provides simulated scenarios, the use of synthetic data is crucial in the context of social navigation. This is primarily because generating a comparable number of situations using only real-world data would be infeasible. Furthermore, scenarios that jeopardise human safety, such as human-robot collisions, cannot be ethically performed in real-world settings.

The robot's movements were generated using two distinct strategies to enhance the diversity of its behaviour. The first strategy employs a machine learning model (see [11]) that outputs the robot's control actions based on a graph representation of the scenario. This model was trained using supervised learning (i.e., containing only examples of appropriate behaviours), which results in unexpected behaviours in situations that do not typically occur when controlled by humans. However, for the creation of *SocNav2*, these behaviours facilitate the generation of a broad range of favourable and unfavourable situations that would not have emerged from random actions. In addition to the samples where the robot's movement was controlled by the machine learning approach, a second set of samples was generated using a joystick for manual control of the robot. This second set was designed to encompass infrequent situations not present in the first set, such as the robot moving backwards to avoid being blocked or stopping to allow people to pass.

Subjects providing scores for *SocNav2* were presented with 4-second sequences and asked to evaluate the robot's behaviour during the final second. The video was shaded during the


(a) First second.


(b) Second second.


(c) Third second.

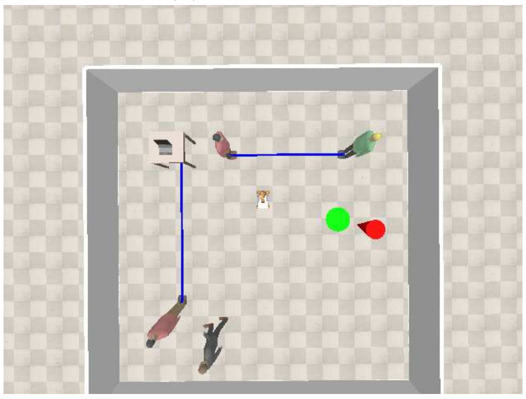

(d) Fourth second

Figure 5.1: A *SocNav2* sequence. The shaded images correspond to the first 3 seconds of the sequence, which are also shown to subjects to provide context. The last image, in Fig. 5.1d, corresponds to the second that the users score. During the whole sequence, the robot is moving forwards.

initial three seconds to facilitate the identification of the specific time slice to be assessed (see Fig. 5.1). The geometrical and relational data of the sequences were stored in JSON files. After watching the video (as many times as necessary), subjects were asked to provide a score for *Q1* and *Q2* based on the reference values provided. Although some guidelines were given, subjects were encouraged to freely express their opinions. The guidelines included:

- Disregard the goal when answering *Q1*. It should only be considered when answering *Q2*.
- The closer the robot gets to people, the more it can be considered disturbing.
- In smaller rooms with a higher number of people, closer distances are more acceptable compared to larger rooms with fewer people.

- The robot must avoid colliding with objects or walls. If it collides, it should receive a score of 0.

	Subject1	**Subject2**	**Subject3**	**Subject4**
Subject1	0.83	0.75	0.80	0.56
Subject2	0.75	0.88	0.85	0.63
Subject3	0.80	0.85	0.88	0.62
Subject4	0.56	0.63	0.62	0.81

Table 5.1: Inter-rater and intra-rater consistency of four subjects for Q1.

	Subject1	**Subject2**	**Subject3**	**Subject4**
Subject1	0.74	0.68	0.72	0.57
Subject2	0.68	0.71	0.74	0.63
Subject3	0.72	0.74	0.76	0.64
Subject4	0.57	0.63	0.64	0.73

Table 5.2: Inter-rater and intra-rater consistency of four subjects for Q2.

Six subjects participated in scoring the dataset, yielding a total of $13,406$ scored samples. This initial set of samples was expanded through data augmentation. Specifically, each scenario was mirrored along the vertical axis, assuming the same scores as in the original scenario. Moreover, each raw and mirrored scenario was rotated by $180°$, with the robot's advance speed sign also being changed. This extension assumes that human discomfort remains unchanged regardless of whether the robot is moving forwards or backwards. Consequently, the data augmentation process resulted in a final dataset comprising $53,600$ samples.

To analyse the consistency of the dataset's scoring, the inter-rater and intra-rater agreements were computed for four subjects using the linearly weighted kappa coefficient [41]. For the inter-rater consistency, common samples scored by each pair of subjects were considered, with a minimum of 609 common samples for which this coefficient was obtained. To measure the intra-rater reliability, each user scored 200 duplicate samples. Tables 5.1 and 5.2 display the inter-rater and intra-rater consistency for the scores of Q1 and Q2, respectively (intra-rater in the diagonal cells, inter-rater in the remaining cells).

As demonstrated in Table 5.1, the intra-rater consistency for Q1 is "almost perfec" according to the scale defined in [102]. The inter-rater agreement for Q1 can be considered substantial in most cases, with only subjects 1 and 4 having a lower agreement, which still falls within the high "moderate" range. Table 5.2 reveals that the consistency for Q2 is generally lower than for Q1. This decrease can be attributed to the question's nature, as subjects may vary significantly in their opinions about how the robot should "efficiently" reach the goal position. Nonetheless, the inter-rater and intra-rater consistency remains "substantial", except for subjects 1 and 4, which are considered high "moderate".

5.3 Video to Graph Transformation

This chapter adopts the strategy developed in the previous chapter for graph generation and incorporates several modifications to account for velocity and trajectory information. This section details the scenario-to-graph transformation process, wherein each scenario consists of a short video, as discussed in Section 5.2. These graphs encode the room's information and are used to generate discomfort scores by *SNGNNv2*, as seen in Section 5.4. Furthermore, these graphs are similar to the ones employed for training *SNGNN-2Dv2*, with some modifications and the addition of a grid of nodes to adapt it for image generation (see Section 5.5.2).

5.3.1 Graph Structure

The input graphs for *SNGNNv2* models are composed of a sequence of three sub-graphs corresponding to three snapshots of the videos shown to the subjects. Each sub-graph (referred to as a 'frame graph') is separated by a one-second interval, with the last graph being the one that users scored. The graph creation process entails two steps. First, each snapshot is transformed into a separate frame graph. Once the three frame graphs in the sequence have been generated, they are merged into a single graph representing the sequence (see Figure 5.2). This temporal connection is established with an edge linking the node in each frame graph to the corresponding node in the subsequent frame graph.

The nodes in the graphs have five types:

- **room (r):** There is one room node per frame graph. It acts as a global node [15] and it is connected to any other node of the graph for that frame. Using a global node

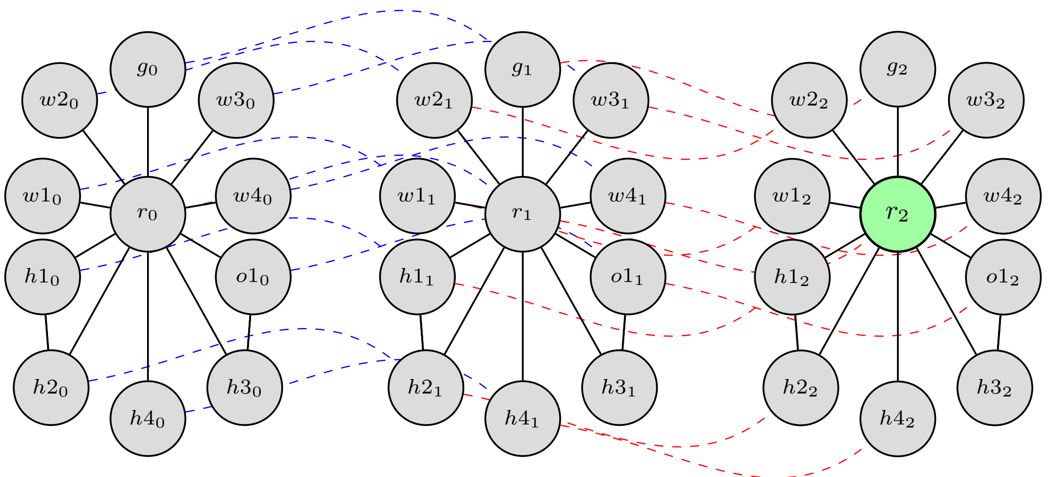


Figure 5.2: Example of how the scenario-to-graph transformation works, based on the scenario depicted in Figure5.1. The sub-indexes of the node names refer to the time frame of each node.

favours communication across the graph and reduces the number of layers required.

- **wall (w):** A node for each of the segments defining the room.
- **goal (g):** Used to represent the position that the robot must reach.
- **object (o):** A node for each object in the scenario.
- **human (h):** A node for each human. Humans might be interacting with objects or other humans.

In the generated graph, there is no node explicitly representing the robot because all node features are in the robot's reference frame (further explained in Section 5.3.2). For every human involved in interactions, two new edges are added between the human and the entity (human or object) they interact with, one in each direction. The graphs also include self-edges for all nodes, and the room node is connected in both directions to the rest of the nodes in the graph. For instance, Figure 5.1 illustrates four frames of a sequence where four humans are in a room with several objects. Two of the humans are interacting with each other, another human is interacting with an object, and the remaining human is moving without interacting with another human or object. Figure 5.2 displays the structure of the resulting graph.

5.3.2 Node and Edge Features

Node feature vectors are constructed by concatenating different sections. The first section is a one-hot encoding for the type of node. The remaining sections are type-specific and are only filled if the node is of the corresponding type, filled with zeros otherwise. The features used in the sections for human, wall, and object nodes are: position, distance to the robot, speed, and orientation, all from the robot's frame of reference. Position and distance are represented in decametres for normalization purposes. Similarly, the orientation is split into sine and cosine, instead of including the angle itself. For wall segments, the position is the centre of the segment, and the orientation is the tangent. Object sections also contain width and height features defining the object's bounding box. The section corresponding to the room symbol is composed of the number of humans in the room and the velocity command given to the robot. Table 5.3 illustrates this layout.

n. one-hot	5 elements (one per node type)				
f. one-hot	3 elements (one per frame graph)				
room	number of humans		adv. speed	rot. speed	
human	pos. (2D)	speed (3D)	orientation (2D)	distance	
object	pos. (2D)	speed (3D)	orientation (2D)	distance	shape (2D)
wall	pos. (2D)	orientation (2D)		distance	
goal	pos. (2D)		distance		

Table 5.3: Structure of the feature vectors of nodes. The first two sections refer to the one-hot encodings that specify the node types and the frame they belong to. Positions (*pos.*) are defined by 2D Euclidean coordinates. Speeds are expressed using 3 dimensions for the linear and angular velocities in the plane. Orientations are given by the corresponding sine and cosine values. All metric values are in the robot's reference frame.

Edge features were implemented differently for the experiments depending on the blocks used. Some GNN blocks such as GAT or GCN, do not support edge features or labels, so no edge information is provided when they are used (see Section 3.2 for more details). R-GCN blocks support edge labels, so a different label is used for each possible type of relation (*e.g.*, human-human, human-room, wall-room). MPNN blocks treat edge information as features not limiting it to identifiers. Therefore, besides containing values identifying the kind of relationship as a one-hot encoding, edge features also include the distance between the two entities being linked when using MPNN blocks.

5.4 Prediction of Discomfort Scores in Dynamic Scenarios, SNGNNv2

This section outlines the procedure for creating the *SNGNNv2* architecture. This model is capable of taking a graph with the information of the environment and output the two scores detailed in Section 5.2. Different GNN models are trained to estimate people's comfort given a scenario and its previous states. GNNs are proposed because the information that social robots can work with is not just a map and a list of people, but a more sophisticated data structure where the entities represented can have different relations among them. For example, social robots could potentially have information about who a human is talking to, where people are looking at, who are friends with whom, or who is the owner of an object in the scenario. Regardless of how this information is acquired, it can be naturally represented using a graph, and GNNs are a particularly well-suited and scalable machine learning approach to work with these graphs.

Based on the assumption that in real-life scenarios, the model can be used with third-party body trackers (e.g., [140] or the models developed in Chapter 6) and path planning systems as demonstrated in 5.5.4, the evaluation of the *SNGNNv2* is performed only against the *SocNav2* dataset presented in Section 5.2, which only focuses on the step of generating scores.

Because all nodes are connected to their corresponding room node, the GNNs were trained to perform backpropagation based on the feature vector of the room node in the last layer. Three GNN blocks were considered in the experiments: the two best-performing GNN blocks for *SNGNN* [118] (*i.e.*, R-GCN and GAT) and MPNN. The reader can refer to Chapter 3 for more information about these networks.

The benchmarking of the different architectures involved 341 training sessions using the *SocNav2* dataset, with $47,598$ samples for training, 643 for evaluation, and 643 for testing. Considering the variability of scenarios, 643 samples were deemed to be a representative sample set size. Hyperparameters were randomly sampled from the range values shown in Table 5.4. Table 5.5 presents the results for the best model of each architecture, detailing the performance on the different dataset splits.

hyperparameter	min	max
max. epochs	1000	
patience	4	
batch size	25	70
hidden units	20	90
attention heads	3	10
number of bases	4	24
learning rate	1e-4	5e-4
weight decay	0.0	1e-6
layers	3	9
dropout	0.0	1e-6
alpha	0.1	0.3

Table 5.4: Ranges of the hyperparameter values sampled. *Attention heads* is only applicable to GAT blocks. *Number of bases* is only applicable to R-GCN blocks.

GNN block	training loss (MSE)	development loss (MSE)	test loss (MSE)
R-GCN	0.017347	0.040098	0.040607
GAT	0.015188	0.037838	0.035818
MPNN	0.025020	**0.036821**	0.035192

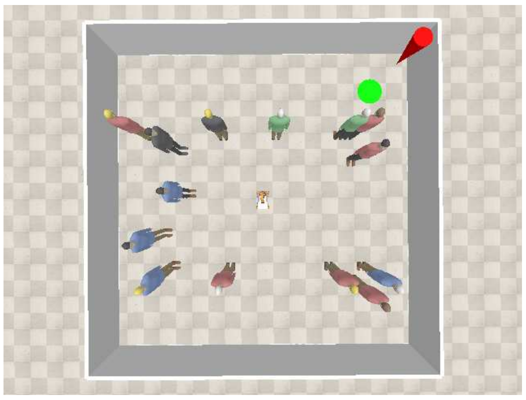
Table 5.5: The 3 GNN blocks tested along with their MSE for SocNav2.

The training results obtained (see Table 5.5) show that MPNN blocks delivered the best results, with a Mean Squared Error (MSE) of 0.036821 for the evaluation dataset. The best model, which was selected based on the MSE on the evaluation split, yielded an MSE of 0.035192 for the test split. The best-performing model was trained with a batch size of 57, a learning rate of 2.5e-4, weight decay regularisation of 1.0e-6 and no dropout. Its **network architecture** is a sequence of **6 MPNN blocks** with 40, 30, 21, 12 and 3 hidden units.

The code to test the resulting GNN model, including the code implementing the scenario-to-graph transformation and the code to train the model suggested, has been published in a public repository as open-source software: `https://github.com/gnns4hri/sngnnv2`.

5.4.1 Visual Experimental Results

To provide an intuitive understanding of the network's output, the outputs for the scenarios shown in Figures 5.3, 5.4, and 5.1 were computed. The model's output was considered for all different robot positions in the room, resulting in a heatmap representation of the network's response for each tested scenario. Note that a query to the network is done for each possible position of the robot. To facilitate the interpretation of each heatmap, the elements present in the scenarios have been drawn over the image using the following representation: oriented blue circles for humans, small green circles for objects, a larger green circle for the goal position, and red lines for interactions. The horizontal and vertical axes of the room's frame of reference are also depicted using black dashed lines, which help differentiate the heatmaps.


(a) Fourteen humans.

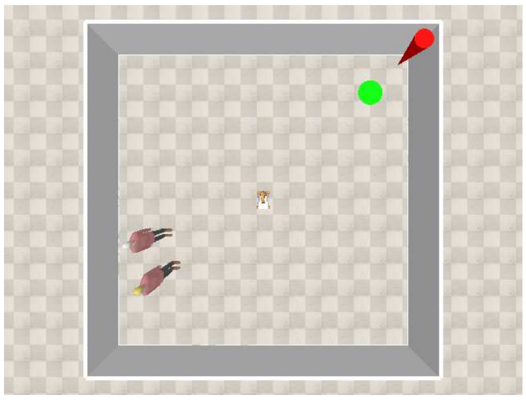
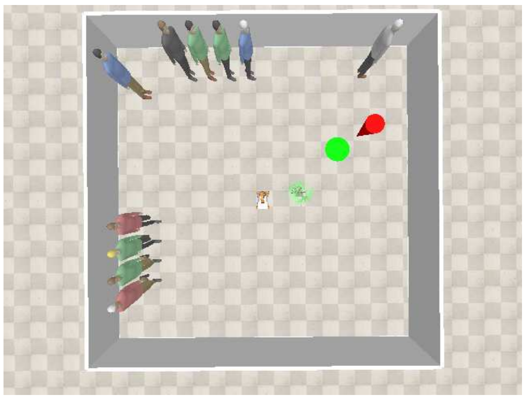

(b) Two humans.

Figure 5.3: Two scenarios containing a different number of people. Results for these scenarios are shown in Figure 5.5.


(a) First frame of the sequence.

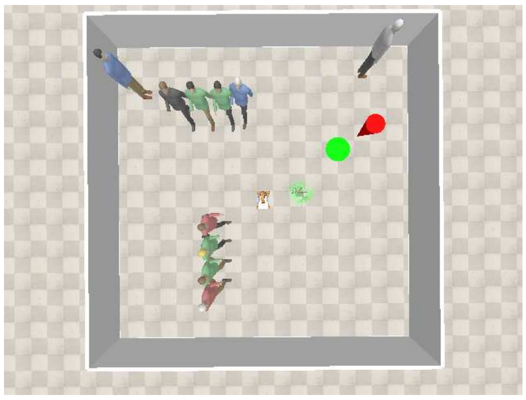
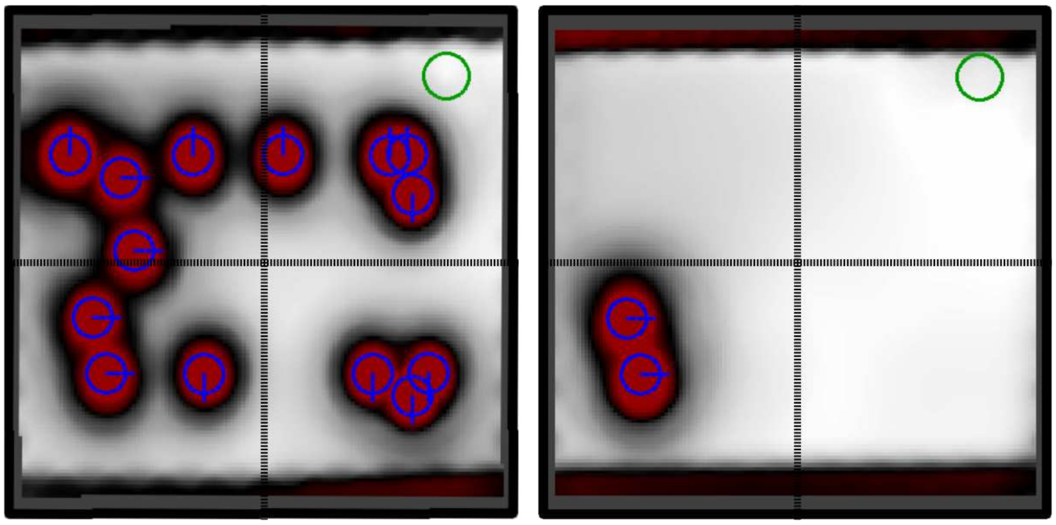

(b) Last frame of the sequence.

Figure 5.4: Scenario with two groups of people walking. Results for these scenarios are shown in Figures 5.6 and 5.7.

Figure 5.5 displays the generated maps for the two scenarios shown in Figure 5.3, considering the network output for *Q1*. Different colours represent the output of the network. Red is used to indicate a value close to 0 (unacceptable situation), while grey tones represent the remaining range of values; darker grey indicates lower values (higher discomfort), and lighter grey represents higher values (socially acceptable). This test demonstrates how the network adapts to variously populated environments. For crowded spaces, such as the one in Figure 5.3a, the discomfort area of the humans narrows compared to scenarios with less dense spaces. For example, the *unacceptable* area of the humans in the bottom left of the room is wider in Figure 5.5b than in Figure 5.5a. Furthermore, the network response increases in positions near the walls when the number of people in the room is high (refer to the goal marked by a green circle in the top right corner of the images). This implies that positions near the room's limits are considered more suitable for crowded environments.

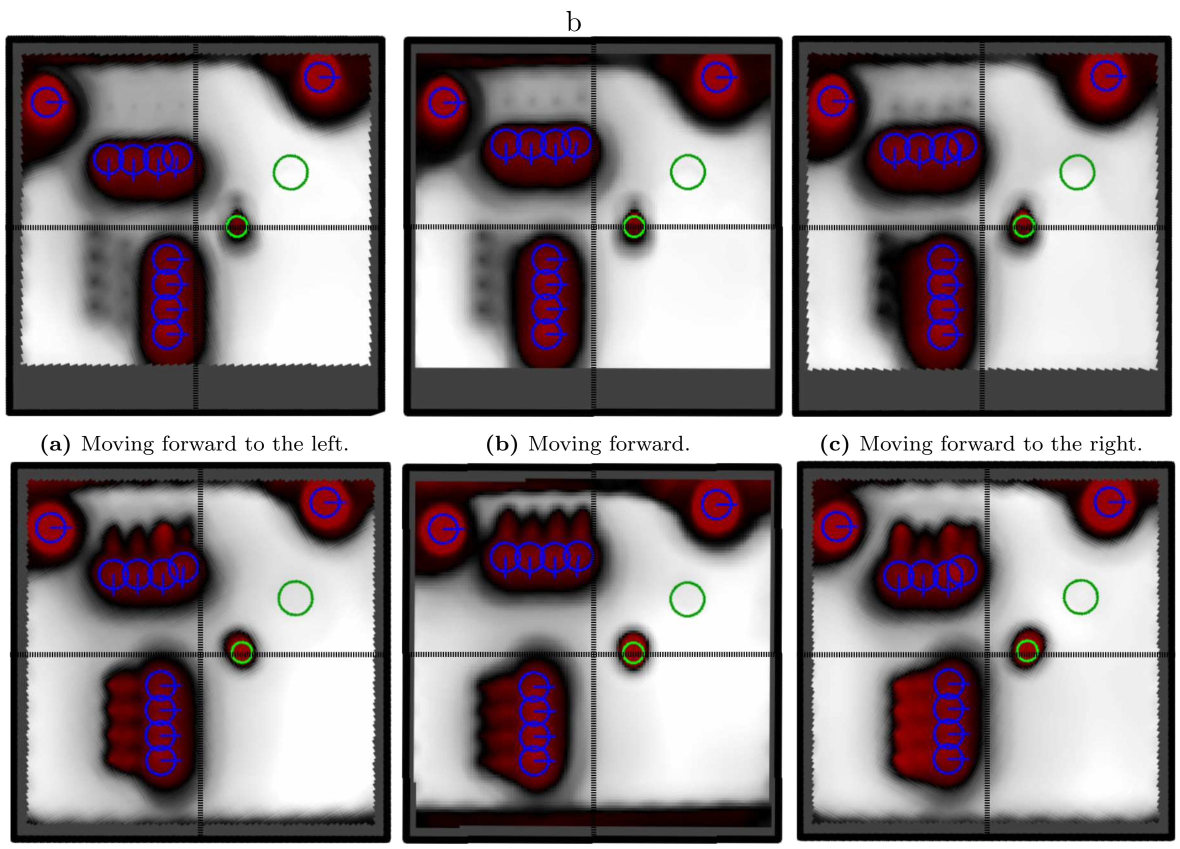


(a) Fourteen humans. **(b)** Two humans.

Figure 5.5: Output of the model for the two scenarios in Figure 5.3. The response of the model is more strict for the case with fewer people.

The scenario in Figure 5.4 has been used to test how the actions of the robot have an influence on the behaviour of the network. Figures. 5.6 and 5.7 show the response of the network for Q1 and Q2, respectively.

In Figure 5.6, the images display the results of *Q1* estimation for different actions of the robot, from bottom to top and left to right: turning left, stopping, turning right, moving forward to the left, moving forward, and moving forward to the right. The *unacceptable* area (red area) for moving people changes based on their motion direction and the robot's actions, whereas for standing humans, this area remains mostly unchanged. When the robot moves forward (Figure 5.6b), the red area for the group of people moving in the opposite direction extends toward their movement direction. In contrast, for the same robot action, the red area for the group of people moving horizontally stays centred at the vertical axis



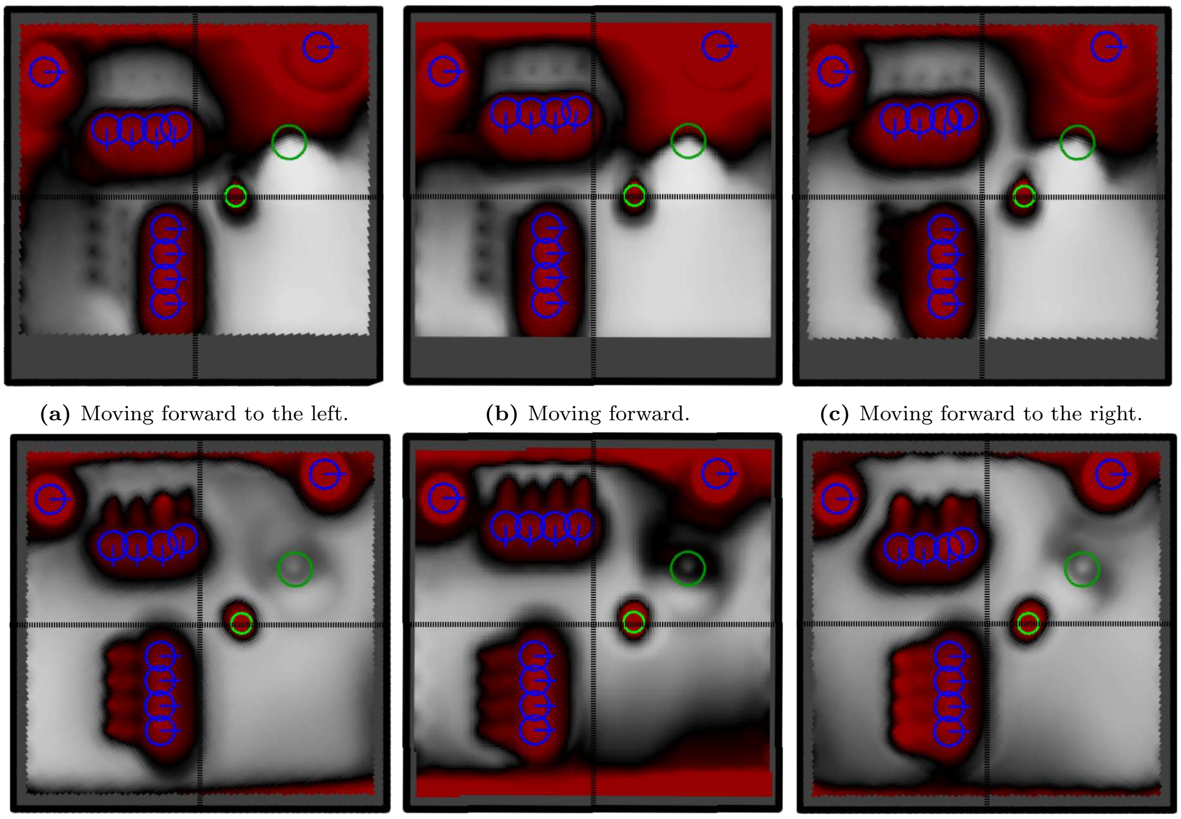
(a) Moving forward to the left. **(b)** Moving forward. **(c)** Moving forward to the right.

(d) Turning left. **(e)** Stopped. **(f)** Turning right.

Figure 5.6: Response of the model for Q1 for the scenario in Figure 5.4 considering different actions of the robot.

position of the humans. For this second group, the unacceptable area extends forwards or backwards when the robot moves to the left (Figure 5.6a) or to the right (Figure 5.6c). When the robot is stopped or turning without translation, the positions with the lowest scores elongate toward the opposite direction of the humans' movement (Figures 5.6d, 5.6e, and 5.6f). These positions correspond to the trajectory followed by the humans during the sequence, indicating that the network's response is consistent with the situation.

The network's response to human presence remains similar for *Q2* (as shown in Figure 5.7), wherein there is an exclusion zone around the humans' positions where the robot is prohibited from entering. However, in this scenario, the positions with low values increase according to the goal position and the action of the robot. For instance, moving forward leaving the goal behind has a very low score. Thus, when the goal is situated behind the robot, the best scoring actions are turning right or left according to the relative position of the goal.

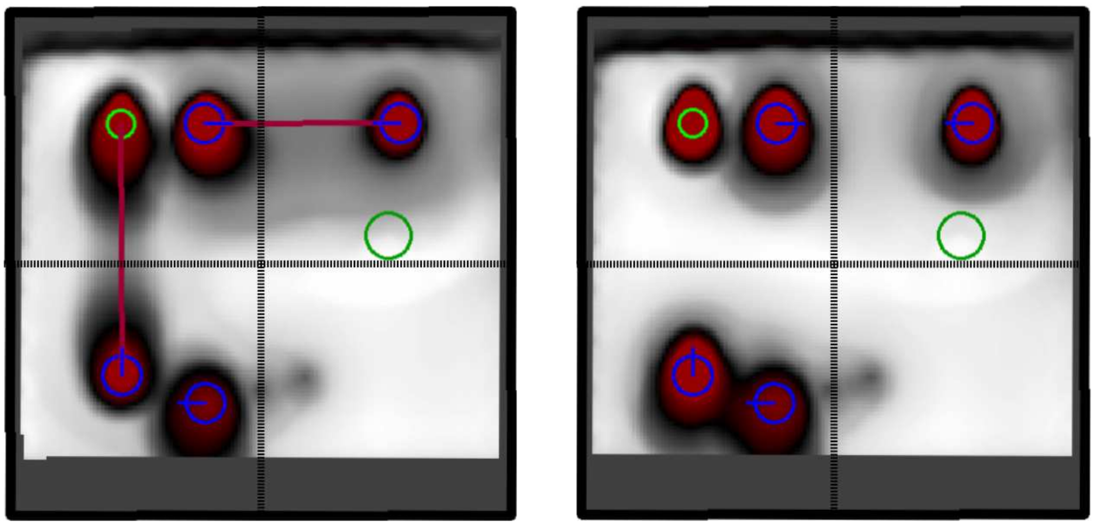


(a) Moving forward to the left. **(b)** Moving forward. **(c)** Moving forward to the right.

(d) Turning left. **(e)** Stopped. **(f)** Turning right.

Figure 5.7: Output of the model for Q2 for the scenario depicted in Figure 5.4 considering different actions of the robot.

To test the network response to potential interactions between humans or humans and objects, the scenario of Figure 5.1 has been used with the robot moving forward. Results for this scenario with and without interactions for *Q1* are shown in Figure 5.8. As can be observed in Figure 5.8a, the interaction between the human and the object produces lower values than the interaction between the two humans. This is consistent with the action of the robot since the human-object interaction takes place in the direction of the robot's movement. Consequently, the interruption caused by the robot's action is more intense than the one produced in the human-human interaction. If no interactions are taking place (Figure 5.8b), the areas between the two humans at the top of the image and the human and the object on the left are considered socially acceptable positions. This demonstrates that the network is properly generalising the different kinds of situations. Another interesting result seen in Figure 5.8b is the different treatment of humans and objects when objects are not being used by humans. Specifically, being close to an object results in a high response, while being close to a human is not considered acceptable. This indicates that the



(a) With interactions. **(b)** Interactions removed.

Figure 5.8: The output of the model for the sequence is depicted in Figure 5.1. Figure 5.8a is the output of the model with the sequence in its original form. Figure 5.8b is the output of the model with the interactions removed. It is apparent that the response of the model for the perpendicular interaction is lower than that of the parallel one. This aligns with the intuition that the robot would be less disturbing if crossing perpendicularly than along the interaction line.

network can distinguish between different types of entities and their interactions, adapting its assessment of social acceptability accordingly.

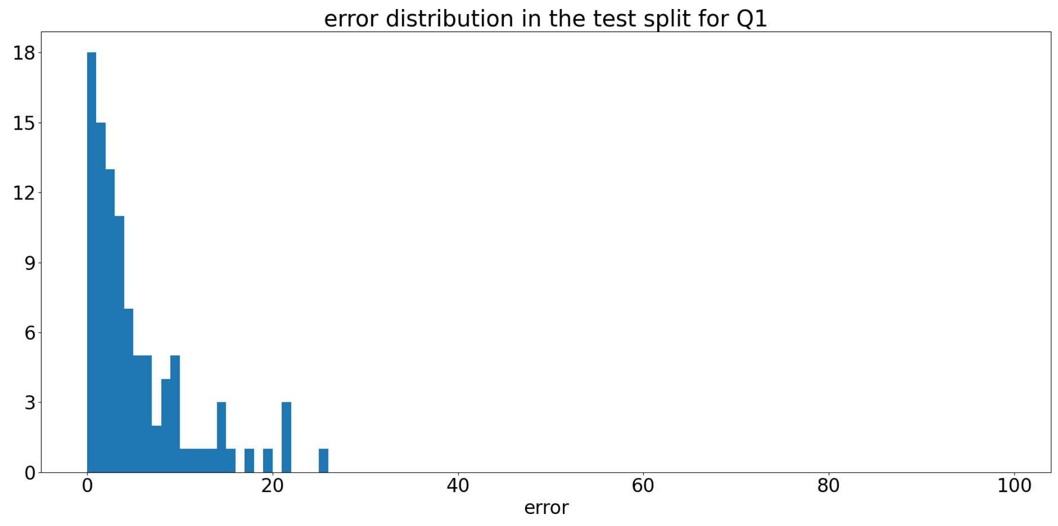


Figure 5.9: Histogram of the absolute error in the test dataset for Q1.

The subjective nature of the scores in the dataset means that there is some level of disagreement even among humans, as human feelings are highly subjective. To compare the performance of the network with human performance, we used a subset of the samples in SocNav2 which was labelled twice by each of the subjects (the same subset used to

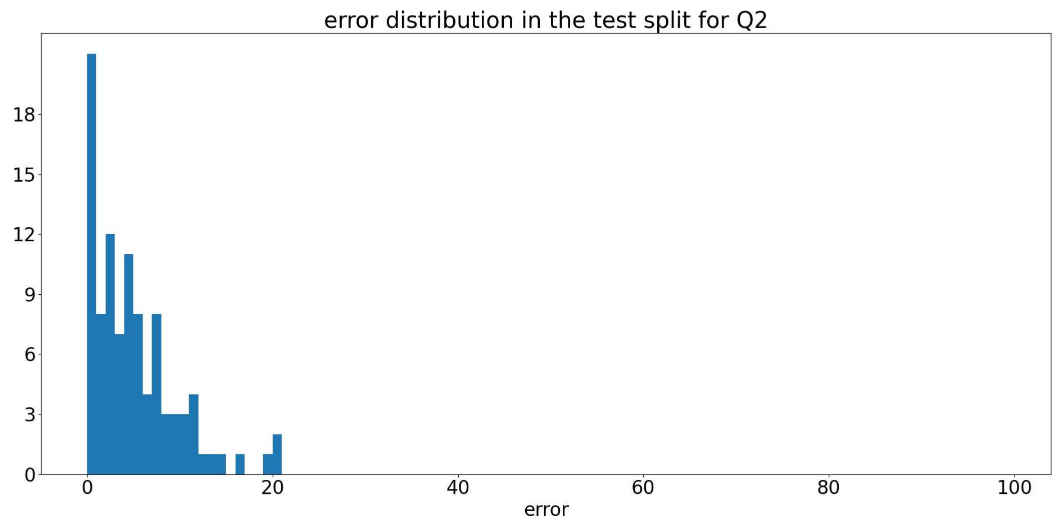


Figure 5.10: Histogram of the absolute error in the test dataset for Q2.

obtain Tables 5.1 and 5.2). Each scenario is annotated twice by each subject to mitigate the subjective discrepancies among measurements and to achieve more stable scores. The mean of the 8 scores provided for each scenario was used as a reference, and the mean squared error (MSE) for each of the participants was computed. The average MSE was 0.036981, which is considered indicative of human accuracy. The network's performance, with an MSE of 0.035192, is close to human accuracy and even slightly better. Figures 5.9 and 5.10 show the histograms of the model's error in the test split of the dataset for *Q1* and *Q2*. In [118] (*SNGNN*), the results were compared (disregarding speed) with [181], achieving a considerably lower MSE (0.022 versus 0.12965). Although the comparison was favourable, it is not entirely fair, as the approaches have slightly different goals. Other researchers are currently working with the datasets used in this chapter and *SocNav1* [119], but there are no published works available for comparison at the time of writing.

5.5 Generation of Maps from Dynamic Data, SNGNN-2Dv2

This section outlines the development and experimentation of *SNGNN-2Dv2*, a model capable of generating two-dimensional disruption maps for dynamic environments from graphs. As discussed in Section 5.4, it is possible to construct these maps by querying *SNGNNv2* for every potential robot position within the room, obtaining the value of each pixel in the final disruption map. Nevertheless, this method is exceedingly time-consuming for real-time applications, averaging 140 seconds per map generation.

Adopting a similar approach to Chapter 4, a dataset comprising graphs and images is employed to train a GNN and CNN combination for expeditious map generation. Section 5.5.1 explains the bootstrapping process for converting scalar dataset labels to 2D image labels using *SNGNNv2*. Subsequently, Section 5.5.2 describes the modification of graphs in Section 5.3 to incorporate a lattice of nodes, facilitating the utilisation of the transpose CNN for generating the final maps. Lastly, Section 5.5.3 outlines the pipeline of the ultimate model, and Section 5.5.4 presents the experimental procedures conducted to evaluate the performance of SNGNN-2Dv2.

5.5.1 Bootstrapping of 2D Dataset from SocNav2

As in Chapter 4, it is necessary to create a dataset with 2D images as labels for training the cost maps generator model in a supervised fashion. Although the data directly acquired from humans is of scalar nature, a two-dimensional dataset can be bootstrapped by generating random scenarios and making multiple queries to the *SNGNNv2* model, one per pixel of the data sample as we saw in Section 5.4. Each pixel of the cost maps is generated by using *SNGNNv2* to estimate the score **Q1** for that position of the robot in the room. Figure 4.3 depicts this sampling process. Note that *Q1* score has been used since it does not account for the goal of the robot. As the main goal of *SNGNN-2Dv2* is to be used for planning, no specific robot movements must be considered. For this reason, to generate the 2D images, the robot remains still during the generation of the graphs used as input to the SNGNNv2 model.

Ultimately, the bootstrapped dataset consists of triplets: small video snippets, corresponding JSON files containing video information as inputs, and matching cost maps as labels. The cost maps, generated from each scenario within the bootstrapped dataset, present a resolution of 150×150. This resolution was selected to balance the quality of the resultant image against the generation time. For training the *SNGNN-2Dv2*, additional data was collected in the same manner as for *SocNav2*, with data augmentation achieved through horizontal and vertical mirroring of the scenarios. The dataset comprises $69,861$ scenarios in total, segregated into $55,711$ training samples, $7,051$ for evaluation, and $7,099$ for testing.

5.5.2 Adding a Grid to the Graph

As mentioned, graph generation constitutes a critical step, given that the training dataset solely provides metric data in *JSON* format from the environment (positions, velocities, orientations), which is unsuitable for direct GNN processing. The graph must encapsulate all pertinent information within the scenario, as well as the interactions among its elements. A well-designed graph is likely to enhance the model's performance and expedite its training. The graph devised in Section 5.3 appears to effectively encode the scenario information, as evidenced by the satisfactory results obtained with *SNGNNv2*. Therefore, *SNGNN-2Dv2* is trained using similar graphs, albeit with several modifications to optimise their generation and ensure their compatibility with the CNN.

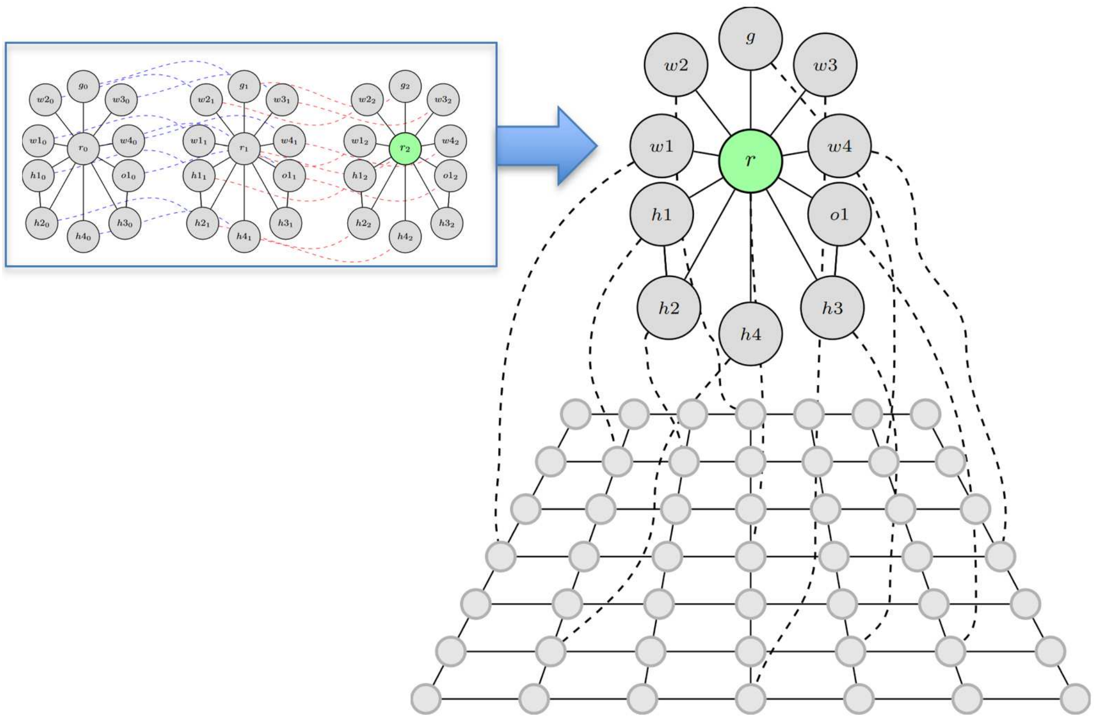


Figure 5.11: Adaptation of Figure 5.2 including the lattice of nodes and merging the 3 temporal graphs into an unique graph.

Firstly, the entity graph, which encodes room information as depicted in Figure 5.2 across three distinct time-frame graphs, is merged into a singular graph. Temporal information is now encoded within the nodes' feature vectors, eliminating the need for temporal connections. This restructuring leads to a simplified graph, enhancing processing speed and efficiency, while also minimising system memory usage. The new graph's topology is portrayed in Figure 5.11, and the nodes' new features, to be detailed later in the text, are shown in Table 5.6.

Secondly, the main alteration to the graph includes adding a lattice of nodes, thereby introducing a new node type: the **grid node** (nodes that constitute the lattice). The grid consists of 19×19 nodes, covering a $13.3m \times 13.3m$ square around the robot. As detailed in Section 4.3.1, the grid serves to encode the 2D scenario information to be used by the transposed CNN to generate the final map. The number of nodes and the area they cover are tunable hyperparameters, balancing performance, computational time, and area coverage. The x, y coordinates of a grid node in row i and column j, from the robot's perspective, are calculated using Equation 4.1.

Finally, the graph resulting from the merged temporal graphs is integrated with the grid by linking each entity node to the nearest grid node, spatially. Each entity node can be connected to multiple grid nodes within a specific radius, leading to the final unified graph depicted in Figure 5.11, following a procedure similar to that in Section 4.3.1.

n. one-hot		6 elements (one per node type)				
f. one-hot		3 elements (max. number of frames)				
grid		pos. (2D)		distance (2D)		
x3	**room**	number of humans		adv. speed	rot. speed	
	human	pos. (2D)	speed (3D)	orientation (2D)	distance (2D)	
	object	pos. (2D)	speed (3D)	orientation (2D)	distance (2D)	shape (2D)
	wall	pos. (2D)	orientation (2D)		distance (2D)	
	goal	pos. (2D)		distance (2D)		

Table 5.6: Update of Table 5.3 including the features for the grid nodes and increasing the one-hot encoding for the node types with one new element (grid-type).

Similar to the graph structure, the node and edge features closely resemble those in Section 5.3, with some modifications. With the union of the three graphs into one, the features of each node representing an entity are likewise merged. The metrics section for each entity node type is triplicated, each corresponding to a distinct frame. There is a one-hot encoding for the number of available frames, ranging from 1 to a maximum of 3. In cases where a frame is unavailable, the fields corresponding to that frame are populated with zeros.

Furthermore, to accommodate the grid node type, an additional element is added to the node type's one-hot encoding, along with a concatenation of grid features. These grid features encompass the 2D position of the grid node and its distance from the robot. Now, all distances are also appended with an additional value—the complement of the normalized distance (1 − distance). As a result, Table 5.6 represents an updated version of Table 5.3, reflecting the graph variant that incorporates the node lattice.

In terms of edge features, new labels are introduced for the grid connections in the R-GCN blocks. Specifically, a label is assigned for each room entity connecting to the grid (e.g., wall-grid, room-grid, human-grid). Additionally, the labels of edges within the grid are differentiated based on the direction of the connection, ensuring an accurate representation of their relative positions (i.e., up, down, left, right). For the MPNN variant, the edges connecting to any grid node retain the same format as the graph in Section 5.2—values identifying the nature of the relationship as a one-hot encoding and the distance between the two connected entities. It is noteworthy that the GNN variation that eventually delivers the best results is a GAT, which does not require these edge features.

5.5.3 Model

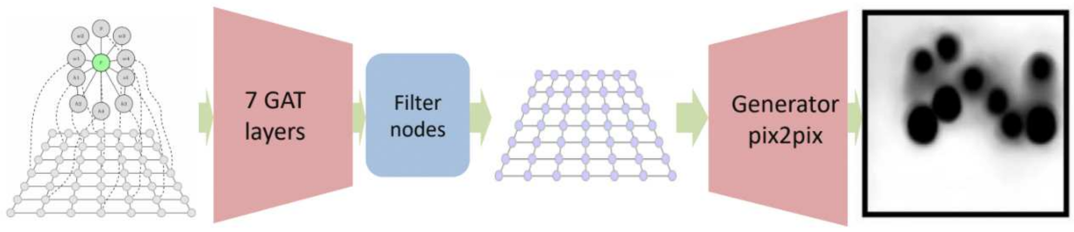


Figure 5.12: Pipeline of the whole *SNGNN-2Dv2* model starting from the input graph and outputting the cost map.

The architecture, depicted in Figure 5.12 has three segments. The first segment is a sequence of 7 GAT blocks. Its output is a graph with the same nodes and structure as the input graph, whose feature vectors have 35 dimensions. The second segment of the architecture filters out nodes that are not part of the grid and restructures the tensor containing the feature vectors of the grid's nodes into a $19 \times 19 \times 35$ tensor so that it can be used as input to the CNN-based network. The third and last segment of the architecture is a U-Net CNN (see Section 2.4.2 for more details) with the same parameters used by the generator of *pix2pix* [81]. The output image after the U-Net has a resolution of 256×256. It is important to note that the label images within the dataset have a resolution of 150×150 pixels. Thus, it is imperative to rescale them to align with the dimensions of the model's output for the loss calculation.

5.5.4 Experiments

This section outlines the results of the experimentation conducted to evaluate *SNGNN-2Dv2* for Human-Aware Navigation (HAN), organized into three subsections: **a)** Precision and Time Efficiency: This segment presents the precision and time efficiency of the trained model on the test set. Visual results of the generated cost maps are also showcased; **b)** Navigation in a real environment: This experiment involves comparing the cost maps generated by *SNGNN-2Dv2* with those created using the model in [181] in a real robot within the ARP Laboratory at Aston University (see Appendix A. The maps are integrated into the ROS navigation stack for HAN, and the results are compared using social metrics; **c)** Comparative Analysis with *pix2pix*: This section provides a comparative analysis with a CNN-based approach, highlighting the advantages of employing a GNN.

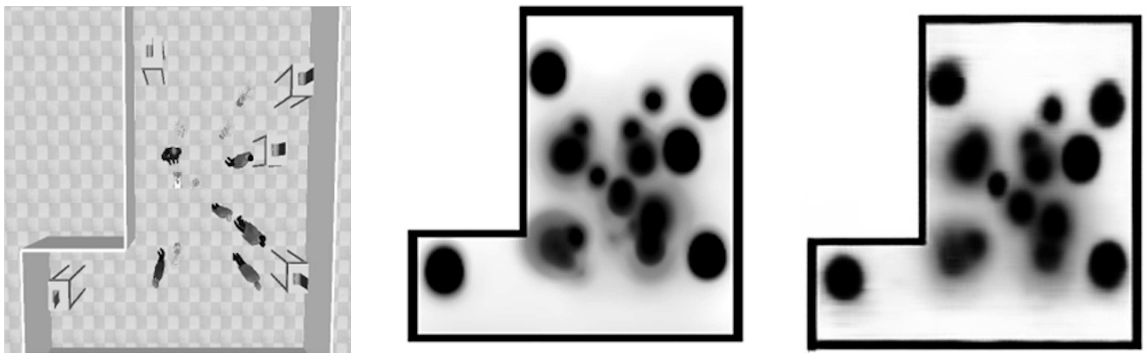
a) Training Results: Time and Accuracy

Figure 5.13 shows the output of *SNGNN-2Dv2* in comparison with the ground truth from the bootstrapped dataset and also shows the scenario from *SONATA*. As the reader can observe, the visual results are almost identical.

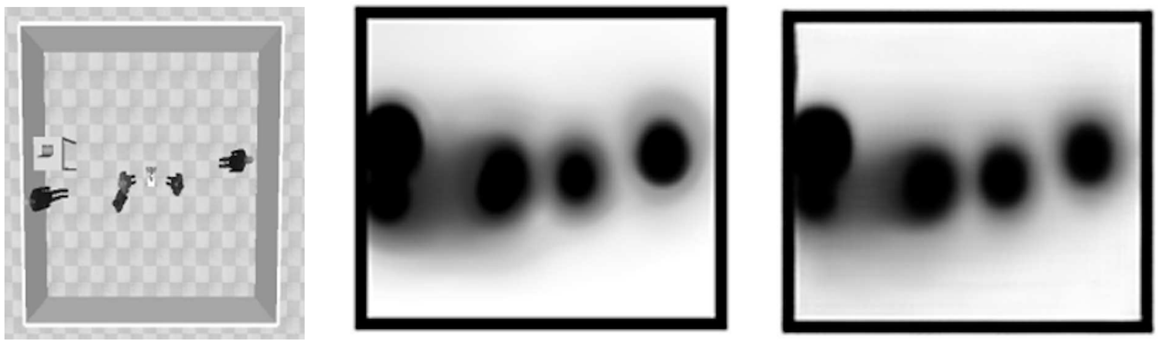
For the optimisation of the hyperparameters, the process of random search ran 67 training tasks using the ranges in Table 5.4. After this process, some of the hyperparameters were manually adjusted to achieve an MSE of **0.001872**, **0.004833** and **0.004750** for the training, development and test datasets, respectively. The average time for a query to the network is of **11** milliseconds in an NVIDIA Jetson AGX Orin [1]. This generation time allows the model to be used in real-time applications as demonstrated in the next experiment (b).

The architecture obtaining the best results is a Graph Attention Network with 7 layers followed by the U-Net of the pix2pix generator [81]. The model reached the best performance on the development dataset after 195 epochs. Table 5.7 shows the hyperparameters used to train the best model:

[1]Specification of NVIDIA Jetson Orin: `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/`



(a) L-shaped scenario.



(b) Squared-shaped scenario

Figure 5.13: Results generated by *SNGNN-2Dv2* (left column) compare with the ground truth (middle column) for two *SONATA* scenarios (right column). The first row shows the results for a L-shaped room and the second row for a square-shaped room.

Hyperparameter	**Value**
Batch size	40
Input channels U-Net	35
Learning rate	5e−5
Activation GAT layer	elu
Final activation GAT	relu
GAT hidden units	$[95, 71, 62, 57, 45, 35]$
GAT heads	$[34, 28, 22, 15, 13, 10]$
Alpha	0.2088642

Table 5.7: Hyperparameters used to train the best model for SNGNN-2Dv2

b) Real Environment Evaluation

To evaluate the maps generated by *SNGNN-2Dv2* for HAN, the robot and environment detailed in Appendix A are utilised. The robot employs the ROS navigation stack with a Timed Elastic Band (TEB) planner [154]. The generated cost maps are published as a ROS topic, to which the robot's navigation system can subscribe and use as a local map for the

TEB. For comparative purposes, the maps generated using GMMs in [181] were also tested.

Human positions and velocities within the room are detected using the 3D pose estimator developed in Chapter 6, in conjunction with the wall-mounted cameras described in Appendix A. A ROS plugin overlays this map onto the map created by the robot's laser, enabling objects to appear in the final map.

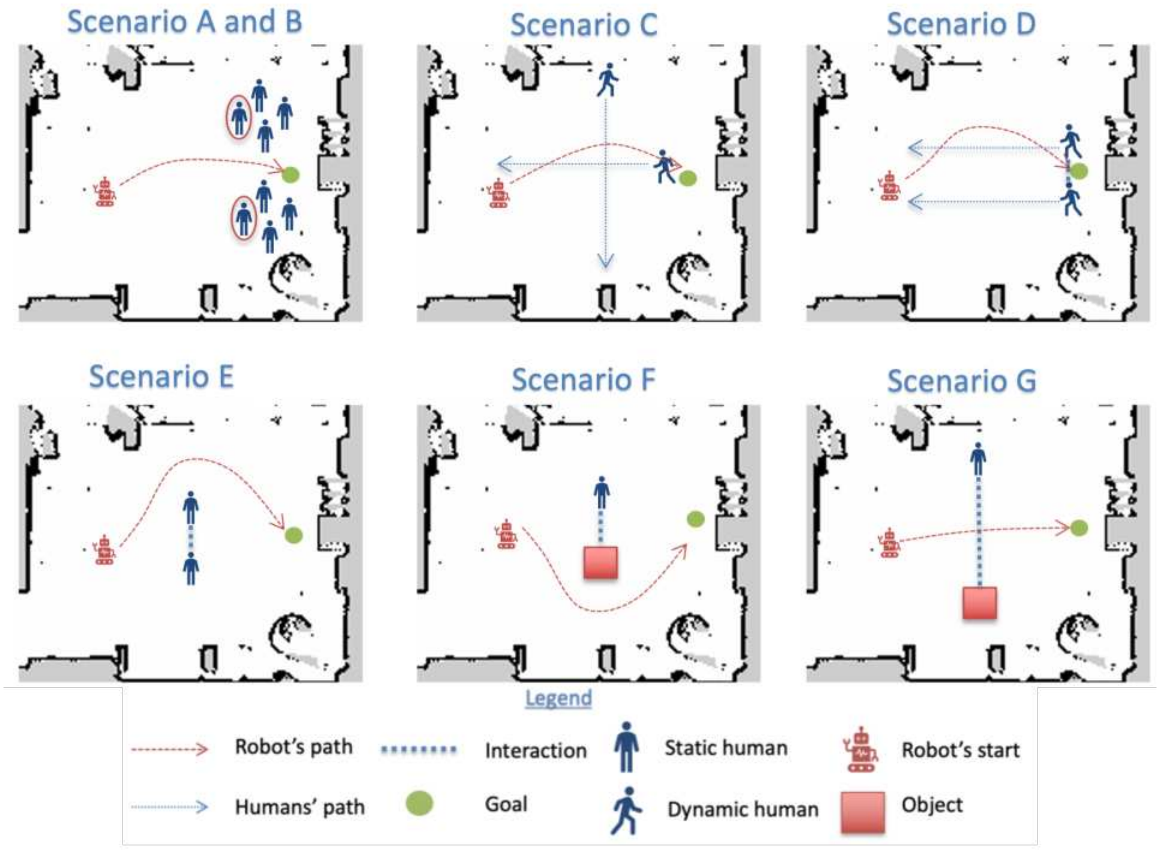


Figure 5.14: Schemes of the different scenarios used to test the maps.

Each map was tested five times across seven different scenarios, illustrated in Figure 5.14. Arguably, these scenarios represent the most prevalent social situations commonly encountered within an indoor space, particularly with a mobile robot. The robot's starting point and goal remained consistent across all experiments, as indicated in the figure. The scenarios consist of:

- **Scenario A (S_A):** This scenario features two groups of three stationary humans, with the groups' centres separated by 2.5 meters. Each group's members are located 0.8 meters from the group centre. The robot's goal is midway between both groups, requiring the robot to traverse between them.

- **Scenario B (S_B):** Identical to S_A, but with an additional human added to each group. The newly added human is highlighted in red in the top-left image of Figure 5.14.

- **Scenario C (S_C):** This scenario involves two moving humans: one moving opposite to the robot and another perpendicularly. Both humans commence movement concurrently with the robot.

- **Scenario D (S_D):** Here, two humans move towards the robot, side by side, with an interaction between them. The robot's goal is at the initial position of the humans.

- **Scenario E (S_E):** This scenario comprises two stationary humans interacting with each other. They stand 1 meter apart, facing each other as if in conversation. The robot should circumnavigate the interaction area to reach the goal on the other side.

- **Scenario F (S_F):** Similar to the previous scenario, but substitutes one person for an object measuring 0.8×0.8 meters. The human faces the object at a distance of 1 meter, interacting with it. Once again, the robot should avoid the interaction area.

- **Scenario G (S_G):** Identical to the previous scenario but with the human and object separated by 3 meters. Given this setup, the robot has insufficient space to bypass the person or object, so it must traverse the interaction area while minimising disruption.

It is important to mention that only scenarios C, D, and E were recorded with real people present. In the other experiments, where the people were stationary, their positions were hard-coded to generate the cost maps, thereby avoiding the small error introduced by the pose estimator in determining their positions. Additionally, it is worth noting that the resolution of the GMM maps had to be reduced by half to ensure real-time responses. The default resolution proved to be too slow for its use with the robot planner.

Figure 5.15 exemplifies scenarios E and D, which were recorded with real people. The right-hand images showcase the robot's path, along with the positions of the individuals tracked by the 3D pose estimator developed in Chapter 6.

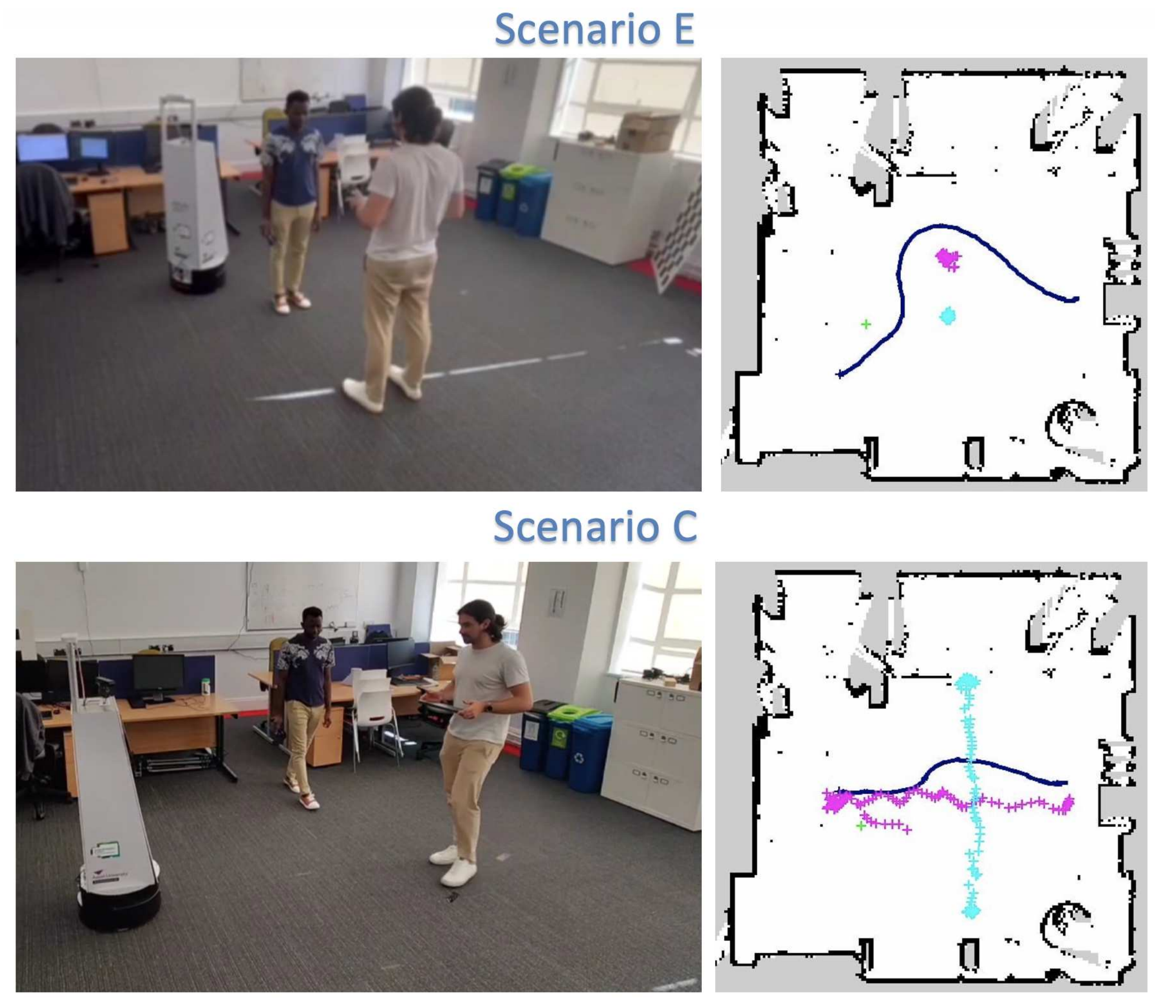


Figure 5.15: Images of the experiments for scenarios C and E. The left-hand images depict actual footage captured during the experiments, while the right-hand images display the recorded positions of the robot and humans. The robot's path is illustrated by a dark blue line, while magenta and cyan colours represent the positions of the two individuals involved in the experiments.

The completion of the experiments was followed by the evaluation of results using the same metrics that are outlined in Section 4.4.4 of the previous chapter, with an additional success rate *sr* metric. Table 5.8 presents the outcomes for all the metrics across all experiments conducted using GMM and *SNGNN-2Dv2* maps. The best metrics values are highlighted in bold.

The analysis of the results leads to a division of the scenarios into three distinct categories. The first category encompasses Scenarios A and B, which involve static groups of people. Here, it is apparent that the majority of the metrics for both types of maps are fairly comparable. However, the robot using *SNGNN-2Dv2* maps reaches the goal more

		S_A	S_B	S_C	S_D	S_E	S_F	S_G
$\tau(s)$	**GMM**	5.211 ± 0.486	4.437 ± 0.244	4.433 ± 0.121	$\mathbf{4.852 \pm 0.005}$	5.042 ± 0.192	$\mathbf{4.134 \pm 1.029}$	$\mathbf{3.383 \pm 0.231}$
	SNGNN-2Dv2	$\mathbf{5.115 \pm 0.946}$	$\mathbf{4.323 \pm 0.189}$	$\mathbf{4.718 \pm 0.192}$	5.298 ± 0.479	$\mathbf{4.837 \pm 0.293}$	4.788 ± 0.381	3.868 ± 0.182
$d_t(m)$	**GMM**	1.134 ± 0.625	1.077 ± 0.026	0.992 ± 0.013	$\mathbf{1.009 \pm 0.008}$	1.051 ± 0.057	$\mathbf{0.967 \pm 0.034}$	1.019 ± 0.044
	SNGNN-2Dv2	$\mathbf{0.967 \pm 0.025}$	$\mathbf{1.01 \pm 0.016}$	$\mathbf{0.991 \pm 0.002}$	1.02 ± 0.02	$\mathbf{0.969 \pm 0.014}$	0.996 ± 0.025	$\mathbf{0.998 \pm 0.027}$
$CHC(rads)$	**GMM**	$\mathbf{3.574 \pm 0.627}$	$\mathbf{2.096 \pm 0.353}$	3.609 ± 0.259	1.801 ± 0.807	4.705 ± 0.448	$\mathbf{3.416 \pm 1.348}$	$\mathbf{1.011 \pm 0.209}$
	SNGNN-2Dv2	4.57 ± 1.696	2.431 ± 0.0435	$\mathbf{3.041 \pm 0.754}$	$\mathbf{2.623 \pm 0.506}$	$\mathbf{4.139 \pm 0.776}$	3.773 ± 1.112	1.21 ± 0.349
$d_{min}(m)$	**GMM**	0.801 ± 0.065	$\mathbf{0.816 \pm 0.019}$	0.585 ± 0.034	0.471 ± 0.083	$\mathbf{1.038 \pm 0.044}$	1.604 ± 0.443	1.688 ± 0.033
	SNGNN-2Dv2	$\mathbf{0.81 \pm 0.03}$	0.731 ± 0.022	$\mathbf{0.771 \pm 0.218}$	$\mathbf{0.607 \pm 0.041}$	0.861 ± 0.048	$\mathbf{2.006 \pm 0.106}$	$\mathbf{1.878 \pm 0.045}$
$si_i(\%)$	**GMM**	0	0	0	2.963 ± 2.963	0	0	0
	SNGNN-2Dv2	0	0	0	**0**	0	0	0
$si_p(\%)$	**GMM**	$\mathbf{59.52 \pm 3.31}$	$\mathbf{64.414 \pm 3.702}$	18.855 ± 1.741	27.164 ± 1.725	$\mathbf{10.155 \pm 3.451}$	15.365 ± 30.729	0
	SNGNN-2Dv2	78.266 ± 4.722	75.216 ± 1.41	$\mathbf{13.799 \pm 5.371}$	$\mathbf{23.342 \pm 1.884}$	37.873 ± 3.345	**0**	0
$si_r(\%)$	**GMM**	0	0	0	0	0	0	100
	SNGNN-2Dv2	0	0	0	0	0	0	100
$sr(\%)$	**GMM**	100	100	80	40	60	100	100
	SNGNN-2Dv2	100	100	80	**100**	**100**	100	100

Table 5.8: Results of the metrics for each of the scenarios comparing *SNGNN-2Dv2* with the GMM model in [181]

quickly, albeit at the expense of closer proximity to humans. In densely populated scenarios, it may be deemed acceptable to be closer to humans due to the lack of available space. This premise is supported by the scores given by participants who contributed to labelling the crowded scenarios in *SocNav2*.

Scenarios C and D form the second category, which includes individuals moving close to the robot. In these cases, the use of *SNGNN-2Dv2* cost maps offers better social metrics, with fewer intrusions into intimate and personal spaces. This implies that the model effectively learns to predict individuals' movements and directions more accurately. Furthermore, the success rate of the *SNGNN-2Dv2* maps is notably higher in Scenario D.

Lastly, Scenarios E, F, and G form the final category, which evaluates the interpretation of human-human and human-object interactions by the maps. In the case of human-human interaction (Scenario E), the robot reaches the goal faster and with fewer heading changes when using *SNGNN-2Dv2* maps. Although it does get slightly closer to the individuals involved, its success rate surpasses that of the GMM maps. In scenarios involving human-object interaction (Scenarios F and G), the robot performance using *SNGNN-2Dv2* maps is similar to that of GMM maps in terms of speed and heading changes. Nevertheless, the robot does not come as close to the human and opts to pass closer to the object instead, thereby highlighting the distinction made between humans and objects in *SNGNN-2Dv2*.

c) Comparison with Pure CNN Based Model

The presented experiment positions the results of cost map generation utilising a CNN-based model, in contrast with the *SNGNN-2Dv2*. The model selected for comparative evaluation is *pix2pix* [81], a highly regarded Generative Adversarial Network (GAN) capable of transposing an image into another. Same to the application in this study, *pix2pix* was trained utilising an identical dataset, achieving a Mean Squared Error (MSE) of **0.0084926** within the test set.

For *pix2pix*, the input volumes are constructed by concatenating three video frames from the dataset along the channel dimension. These frames have a temporal difference of one second, mirroring the method used to construct the graph for the *SNGNN-2Dv2*.

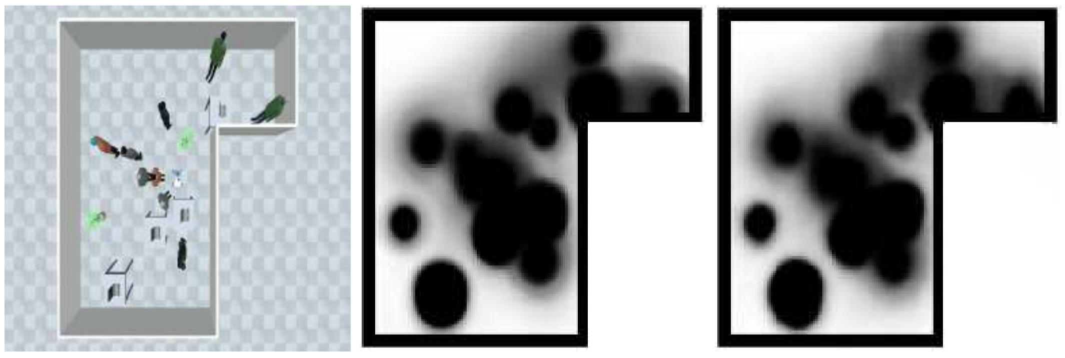

(a) Input **(b)** Ground truth **(c)** Output pix2pix

Figure 5.16: Pix2pix results 1

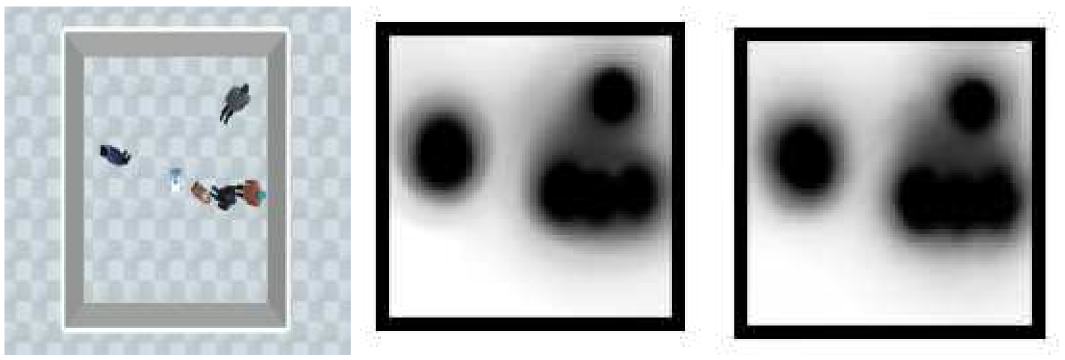

(a) Input **(b)** Ground truth **(c)** Output pix2pix

Figure 5.17: Pix2pix results 2

Figures 5.16 and 5.17 provide a visual comparison of the *pix2pix* results alongside the corresponding ground truth used for training, including a frame from the source video on the left-hand side. As the reader can observe, the outcomes bear striking visual similarity, as well as closely align with the results generated by the proposed model *SNGNN-2Dv2* as illustrated in Figure 5.13. Nonetheless, as previously highlighted, the *SNGNN-2Dv2* model asserts two primary advantages over CNN-based models: superior representation of entity interactions within the environment and an added layer of abstraction that renders the model scenario-agnostic. These benefits are demonstrated across two different experiments.

In the initial experiment, *pix2pix* is tasked with generating cost maps for two scenarios in which two individuals are interacting. The resultant figures can be found in Figures 5.18 and 5.19. Evidently, *pix2pix* lacks any information within the input regarding the occurring interaction. Consequently, it fails to accurately represent the interaction area when compared with our model. This shortcoming is notably observable in Figure 5.18, wherein *pix2pix* incorrectly models interaction areas where no interaction exists and vice-versa.

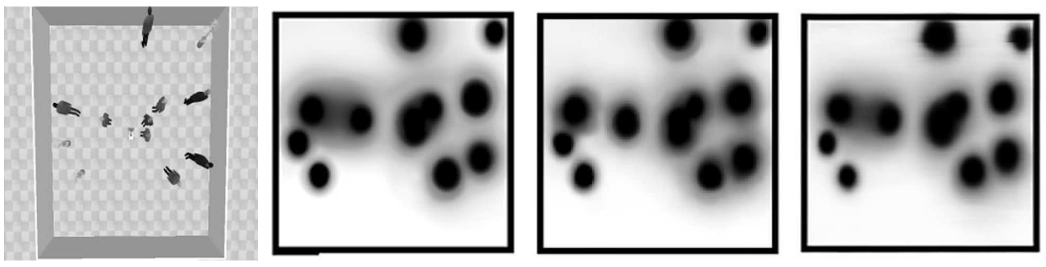

(a) Input **(b)** Ground truth **(c)** Output pix2pix **(d)** Output SNGNN-2Dv2

Figure 5.18: Pix2pix results with interaction 1

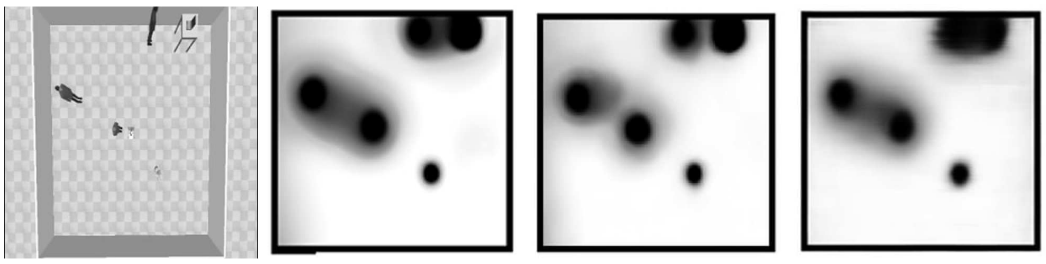
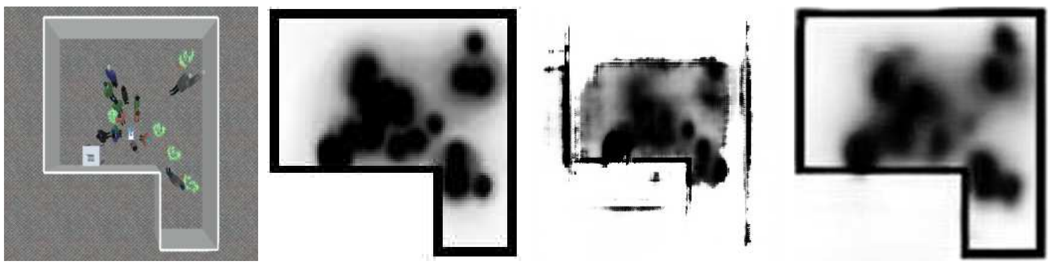

(a) Input **(b)** Ground truth **(c)** Output pix2pix **(d)** Output SNGNN-2Dv2

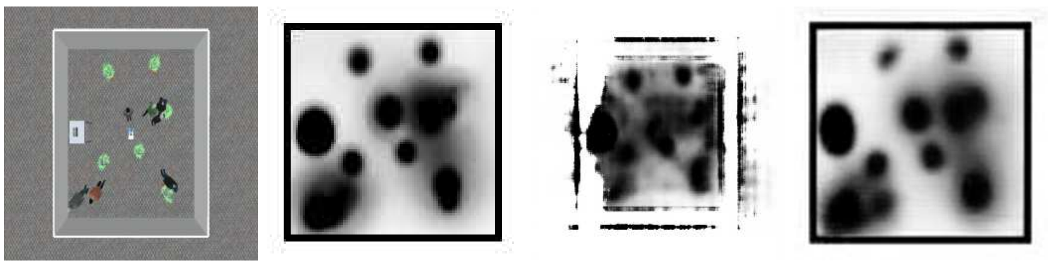
Figure 5.19: Pix2pix results with interaction 2

In the final experiment, alterations were made to the texture and colour of the room floor within the video. Figures 5.20 and 5.21 exhibit the outcomes for two distinct examples. In this case, the output image produced by *pix2pix* is nearly unidentifiable, given its heavy reliance on visual features. Contrarily, the output generated by *SNGNN-2Dv2* remains unaffected by the transformation in the floor pattern.



(a) Input **(b)** Ground truth **(c)** Output pix2pix **(d)** Output SNGNN-2Dv2

Figure 5.20: Pix2pix results with different floor 1.



(a) Input **(b)** Ground truth **(c)** Output pix2pix **(d)** Output SNGNN-2Dv2

Figure 5.21: Pix2pix results with different floor 2.

5.6 Conclusions

To the best of my knowledge, all works modelling discomfort around robots disregards information such as explicit interactions, trajectories or speed. This chapter tackled these issues with a specific scenario-to-graph transformation and a graph neural network architecture, *SNGNNv2*, composed of 6 MPNN blocks.

The results obtained are close to human accuracy and improve those in [118] not only in terms of MSE but also in terms of the features considered (*i.e.*, the trajectory and speed of the robot and the humans). The results confirm that the discomfort is only skewed to the front of the humans when there is movement involved, which was initially hypothesised in [118]. The results also show that: **a)** the model adapts to a variable density of humans (see Fig. 5.5); **b)** static humans are considered more carefully since they are more likely to move; and **c)** the model can consider the interactions which have been given explicitly.

The development of *SNGNNv2* has facilitated the creation of a bootstrap dataset that trains a model, *SNGNN-2Dv2*, to generate cost maps directly from graphs. As demonstrated in Section 5.5.4, this model can be applied to produce maps for real-world human-aware navigation with a robot, achieving superior results compared to the GMM maps generated by the model in [181]. Furthermore, *SNGNN-2Dv2* remains unaffected by changes in the scenario's appearance and captures human-human and human-object interactions, which is a marked advantage over a CNN-based model that relies on raw image inputs. This advantage is underscored in Section 5.5.4.

Future works point to user profiling and personalisation, as well as considering the activity of the humans and their gaze as done in works such as [35]. Also, an ongoing line of research explores ways of linking the output of *SNGNNv2* (questions Q1 and Q2) to driving control of the robot. *SNGNNv2* and *SNGNN-2Dv2* could also benefit from harnessing contextual cues extracted through commonsense reasoning and knowledge acquisition frameworks, akin to the model proposed by Sridharan et al. [165]. This approach can offer the advantage of reduced computational load compared to deep learning methods for scene understanding. An end-to-end solution is a possibility but complicates the acquisition of labelled examples and the modulation of the final control action. An interesting alternative would be to use the output of the GNN as an additional restriction to be fulfilled by a Model Predictive Controller [128].

Chapter 6

Human Pose Estimation from Camera Sensors

Part of the work presented in this chapter has been adapted from the following publications:

[145] **Rodriguez-Criado, Daniel**, P. Bachiller, P. Bustos, G. Vogiatzis, and L. J. Manso. Multi-camera torso pose estimation using graph neural networks. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 827–832. IEEE, 2020.

Part of the work in this chapter is also under review at the *International Journal of Computer Vision* with the title “Multi-person 3D Pose Estimation from Unlabelled Data”.

Human detection, pose modelling and orientation detection have a plethora of applications, including video surveillance [188, 56, 44], assisted living [27], and autonomous vehicles [54]. In addition to any direct application, it is also the basis of trajectory prediction [160, 169], interaction detection, and gesture recognition [3]. Extensive research efforts have been made with different technologies such as LiDAR technology [200, 171], RGB cameras [177], and RGBD cameras [24, 202]. The number and significance of applications make this field highly impactful, thus motivating the work presented in this chapter. This chapter introduces two distinct approaches for pose estimation: the first one focuses on human position in the 2D floor plane and orientation estimation using supervised learning (see Section 6.2), while the second estimates full 3D human pose from unlabelled data (see Section 6.3). The second system is employed in the experiments in Section 5.5.4 of the previous chapter to detect people in the environment; these positions are then used by *SNGNN-2Dv2* to generate the cost maps. It is worth noting that both systems presented in this chapter operate in multi-camera and multi-person environments taking as input raw images from the cameras.

Numerous usability, cost and operational requirements can be anticipated for a Human Pose Estimator (HPE), as well as for estimating people's position and orientation. For instance, most applications require support for multiple individuals. Moreover, except for a few niche cases, HPEs must accommodate occluded body parts. The majority of applications would also benefit from limiting sensors to RGB cameras, avoiding the need for RGBD or other costlier hardware. An ideal HPE would exploit the available context to provide 3-dimensional data for all keypoints, even when not all of them are visible. Another desirable feature for any learning-based HPE would be not to require a labelled dataset when implemented in a new space, as these datasets can be expensive and time-consuming to compile. The HPE models developed in this chapter are designed with these crucial considerations in mind, aiming to incorporate these desired features.

RGB-based multi-human and multi-view estimation applications typically employ a three-step pipeline: a) human detection and 2D pose estimation in images using, for example, a Convolutional Neural Network (CNN); b) identification of correspondences across different views of the people detected in the previous step; and c) estimation of 3D poses, positions, or orientations for each individual based on the image coordinates of their keypoints across various views. Both contributions presented in this chapter build upon publicly available pose detectors, such as those in [1, 26], for the first step of the pipeline. The system introduced in Section 6.2 proposes a novel solution for the third step, while the system in Section 6.3 designs new solutions for the second and third steps. It is crucial to emphasise that the systems developed in this research can be integrated with any third-party 2D detector.

Regarding the second step, which involves associating 2D poses corresponding to the same person across different images, the literature addresses the problem using both appearance and geometric cues. Examples include employing epipolar geometry to assign a cost to each detected pose [22] or embedding appearance features using a pre-trained model to provide affinity scores between bounding boxes [46]. The system developed in Section 6.2 approaches this step in a manner similar to existing literature, relying on geometric cues and using the Bhattacharyya distance between detections in consecutive frames for matching. However, the system in Section 6.3 introduces a new data-driven solution for matching different views from unlabelled data. Given the desired multi-person support and the irrelevance of the order in which people are detected, a Graph Neural Networks (GNNs) architecture is chosen to match people's views, as they are order-invariant and can handle a variable number of input nodes.

Traditionally, the final step, 3D pose estimation, has been performed using triangulation or pictorial structure models. The main limitation of these classic approaches arises from their inability to predict occluded parts, as these methods cannot estimate positions for keypoints that are occluded in many or all views. To overcome these limitations, both systems presented in this chapter utilise a learning-based approach for this step. Arguably, an artificial neural network can learn to *hallucinate* the occluded parts of the body even when they are not visible. This is based on the intuition that the network should be able to exploit contextual information from the remaining keypoints and the existing views, if available. For instance, a network could learn to implicitly internalise the proportions of the human body and its bilateral symmetry. Therefore, if the keypoint for the left elbow is not visible from any camera, the network, by knowing the position of the wrist and the average proportions of a human forearm (or the length of the opposite forearm), could estimate the position of the elbow. Efficiently embedding these complex yet helpful biases would be highly challenging in non-data-driven approaches.

A significant limitation of most current data-driven solutions, particularly those that provide multi-camera support, is the necessity of annotating datasets to train the models in a supervised fashion. It is worth noting that multi-camera datasets are specific to the relative positions of the cameras, rendering the datasets scenario-specific. Consequently, to use the corresponding approaches, an annotated dataset must be compiled for each scenario, which is time-consuming and requires expensive tracking systems. Moreover, the usefulness of a vision-based pose-estimation method would be questionable if a separate system capable of providing ground truth is available. To alleviate this problem, the data gathered in the first project is a mix of labelled and simulated data. Simulation enables the rapid and straightforward generation of new data points. It is the second project that truly introduces a solution to fully avoid the need for annotated datasets. The models in this project are trained in a self-supervised manner. The method is explained in detail in Section 6.3.1.

In summary, the specific **contribution of the first project** is the estimation of human torso 3D pose and its orientation using a GNN from raw images. To the best of my knowledge, there are no previous GNN models to predict pedestrian orientation. The **contributions of the second project** are twofold:

- A solution for matching different 2D poses from multiple cameras using a GNN, allowing for a variable number of people in the scenario.
- A model that infers the 3D keypoints of the detected humans using self-supervised learning by minimising the difference between the 2D detected keypoints' coordinates and those of the estimated poses' re-projections. Furthermore, as demonstrated in the experiments, the system can be applied to mobile robots without retraining for different scenarios, provided only onboard cameras are used.

The next section introduces the most relevant 2D human pose detectors and reviews the current state of the art in position and orientation detection and 3D pose estimation. The remainder of the chapter is divided into two sections containing the two distinct mentioned projects. Section 6.2 details the project for the estimation of human torso positions and orientations, and Section 6.3 introduces the 3D HPE system trained with unlabelled data. Lastly, Section 6.4 summarizes the conclusions of the results obtained in these two projects. It highlights the key findings, contributions, and potential future directions for research in the area of human pose estimation using multi-camera systems and unlabelled data.

6.1 Related Work

This section reviews the leading literature on human position and orientation detection as well as 3D Human Pose Estimation. It begins by briefly covering the most popular 2D detectors, as they are leveraged in many 3D estimation models. It is worth noting that works using RGB-D sensors (with depth channel) are not discussed since the work in this chapter focuses on RGB cameras only, which offers the advantage of a considerable reduction in equipment cost.

6.1.1 2D Pose Detectors

2D human pose estimators yield image coordinates of human anatomical keypoints in an image for every detected person. Recent advancements in deep learning have led to a significant improvement in the performance and accuracy of these models, surpassing the previous approaches that relied on probabilistic and hand-crafted features [32]. Most of these learning-based models [1, 95, 82, 174] rely on Convolutional Neural Networks. Among the vast amount of 2D pose estimators, OpenPose [26] is one of the most popular. It leverages part affinity fields for human parts association using a bottom-up approach. A similar approach is followed by OpenPifPaf [97] and trt-pose[1]. Another widely known 2D pose detector is HRNet [168], which can maintain high-resolution representations through the detection process, claiming higher accuracy and spatial precision. One of the most popular datasets used for training and evaluating these 2D models is COCO [114], containing more than 100.000 annotated images.

6.1.2 Position and Orientation Detection

Accurate **localisation and orientation estimation** are also crucial for human-aware navigation, as demonstrated in the previous chapter. For instance, the orientation of pedestrians' velocity vectors is used in [120] to enable a robot to navigate crowded environments while adhering to constraints defined by proxemics. In this case, the orientation vector is assumed to be parallel to the movement. Although this is often the case, such a strategy becomes noisy when people move slowly. The orientation of humans' joints is also used in [25] to classify motion patterns in human activities. A total of 25 keypoints of the human

[1]Project URL: `https://github.com/NVIDIA-AI-IOT/trt_pose`

body are used to create a feature vector that is input to an SVM to classify motion types (e.g., jump, squat, run).

Although a significant number of exceptions exist (e.g.,[161, 184, 108]), orientation and other social cues are typically obtained using the aforementioned three-stage pipeline for multi-camera and multi-person setups. The final stage algorithm is often implemented employing elementary trigonometry, taking into account the coordinates of the shoulders or hips[56]. For example, in [39], the calculation is performed using the cross-product of vectors extending from the head to the right and left hips, respectively.

In order to address the limitations of handcrafted equations when dealing with missing, noisy, and redundant data, some studies adopt a machine learning-based approach. For instance, both [161] and [184] utilise Histograms of Oriented Gradients (HOGs). In [161], RGB-D HOGs were employed to provide discrete angle estimations. The research presented in [159] also offers orientation estimation based on a set of angle ranges using HOGs for RGB images only. This method generates feature vectors that are incorporated in logistic regression, enhancing the results obtained by [161]. In [184], a similar approach is conducted, also using discrete angles. While achieving high precision of less than $< 0.1°$, this precision is attributed to the subsequent use of a Kalman filter to obtain static orientation. The study in [108] employs RGB-D and IR images with IR trackers to train a single-camera Convolutional Neural Network (CNN), providing continuous angle estimation and attaining a mean absolute error of approximately $6°$.

The accuracy of the work conducted in Section 6.2 for position and orientation estimation is slightly below that of [108]. Nonetheless, the proposed methodology in this chapter not only estimates orientation but also determines the 3D coordinates of the torsos without necessitating the use of relatively costly RGB-D cameras.

6.1.3 Full 3D Human Pose Estimation

Regarding the **3D pose estimation** problem, fuelled by the remarkable progress in 2D estimations, numerous studies have attempted to leverage these models to estimate 3D poses from 2D points [186]. Many of these approaches derive 3D human poses from monocular views [111, 148, 121, 138, 124]. However, they are constrained by the inevitable reality that monocular depth estimation is an ill-posed problem, as a single 2D projection can yield multiple potential 3D poses. **Multi-camera** systems can significantly reduce this ambiguity and enhance robustness against occlusions and noise. Nevertheless, multi-view human pose estimation involving **multiple people** introduces the challenge of matching each individual's set of keypoints among images from different cameras. Previous research has tackled this issue using algorithms based on appearance and geometric information [46, 18]. Dong et al. [46] construct affinity matrices based on the appearance between two views and utilise them as input to their model to infer the correspondence matrix.

Upon resolving the cross-view correspondences, there are several techniques to merge the information from the different views to extract the 3D pose. Classical approaches predominantly rely on epipolar geometry, triangulating the 3D points from 2D points [8, 18, 92]. The pictorial structure paradigm was extended to 3D in [17] to address multi-human pose detection. However, the model does not identify complete skeletons in cases of occlusion, and it assumes prior knowledge of the number of people in the scene, which is unrealistic [177]. Other studies confront the problem using prediction models based on **deep learning** and CNNs [177, 201, 145]. For instance, VoxelPose [177] discretises the 3D space into small cubes called voxels. Utilising this representation, the 2D heatmaps detected from all views are projected into a shared 3D space, and two 3D convolutional models are applied. The first model generates detection proposals for each individual, while the second estimates the positions of the keypoints for each proposal. This method avoids establishing cross-view correspondence based on low-quality 2D poses. Ye et al.[201] introduce an accelerated version of VoxelPose that circumvents the use of 3D convolutions, albeit with slightly inferior results. Initially, they re-project the aggregated feature volume, acquired similarly to[177], to the ground plane (xy) by implementing max-pooling along the z-axis. Subsequently, they employ a 2D-CNN network over the xy-plane to identify individuals and generate a 1D feature vector in the z-axis for each detection. Lastly, they apply a 1D-CNN to that vector to obtain the final 3D pose estimation. These modifications allow their model to achieve results approximately ten times faster without compromising accuracy. Another noteworthy study by Lin et al.[113] utilises a plane sweep stereo technique to simultaneously tackle the challenges of multi-view fusion and 3D pose estimation. It is important to mention that all these models are camera-configuration specific and need scenario-specific datasets with

Qualitative features	Multi-camera	Multi-person	Self-supervised
Tu et al. [177]	✓	✓	✗
Ye et al. [201]	✓	✓	✗
Wo et al. [194]	✓	✓	✗
Lin et al. [113]	✓	✓	✗
Biswas et al. [20]	✗	✗	✓
Kundu et al. [100]	✗	✗	✓
Srivastav et al. [166]	✗	✓	✓
Model in Section 6.3	✓	✓	✓

Table 6.1: Qualitative comparison with literature

accurate annotations. Addressing this second issue constitutes the **primary objective** of the work presented in Section 6.3. The scarcity of such datasets can be attributed mainly to the expensive equipment required and the need for a controlled environment to record data. Examples include the Human3.6M dataset[80], featuring over 10,000 annotations from 1,000 images, and the CMU Panoptic dataset [84], containing 5.5 hours of video from various angles and 1.5 million 3D annotated skeletons.

Kocabas et al. [92] introduce an approach that is conceptually related but methodologically distinct from the work under consideration. They employ self-supervised learning for 3D human pose estimation by calculating the loss using epipolar geometry from multiple cameras' 2D pose estimations. However, their method only provides the 3D pose for a single individual in the scenario, using a combination of two CNNs.

It is worth mentioning multiple works in the literature addressing the problem with the use of GNNs. For instance, works such as [197, 74] achieve promising results from monocular views. Wu et al. [194] propose a solution for multi-view and multi-person 3D estimation using supervised learning. They utilise a GNN for both cross-view correspondences (an approach similar to the one presented in 6.3) and the final 3D pose estimation. The graphs are constructed by converting each detected keypoint into a graph node and using the natural connections in the body to create the graph edges. The GNN then applies a regression to the node features to determine the 3D coordinates of the body joints. The main limitation is that the training of these networks requires datasets with accurate 3D ground truth annotations.

Due to the significant advantages of avoiding 3D annotations (*i.e.*, making datasets more affordable in terms of cost and effort, which facilitates the collection of larger datasets), self or semi-supervised learning methods have been proposed in numerous studies [47, 138, 143, 92, 20, 100, 166]. These methods either utilise only a single view or are limited to estimating the 3D pose of a single individual. Thus, to the best of my knowledge, the work in Section 6.3 is the first multi-camera 3D Human Pose Estimation method that supports multiple individuals and does not require ground-truth data. Table 6.1 presents a qualitative comparison of the most relevant and recent works in the literature, demonstrating that the model developed in Section 6.3 is the only one that satisfies all three characteristics.

6.2 Multi-camera Torso Pose Estimation

This section addresses the problem of estimating a person's torso pose from a set of cameras. In this case, the pose is defined as the person's torso position on the floor plane and their orientation with respect to the vertical axis: (x, y, α). The system should be able to cover spaces large enough to require multiple cameras attached to the walls with overlapping fields of view.

This project employs both real and synthetic training data to generate a large dataset in a relatively short time, saving a considerable amount of resources. The hypothesis is that the data augmentation coming from simulation will contribute to improving results compared to training solely with real data. The first row of Figure 6.1 displays images of the real environment, while the second row shows a representation in CoppeliaSim [149] of three views of the environment where the system has been tested. Both systems have the same camera setup with identical calibration, using an AprilTag in the same relative position as a reference for the origin of coordinates.

As previously mentioned, the processing pipeline illustrated in Figure 6.2 is comprised of three key phases. Initially, images are acquired and processed using OpenPifPaf [96]. It should be noted that any other 2D detector can be utilised during this step. The output data from this phase, a set of detected skeletons from different cameras, is passed on to the subsequent phase, where skeletons corresponding to the same person are matched and grouped. These groups are then supplied to a GNN, which generates the final output. The

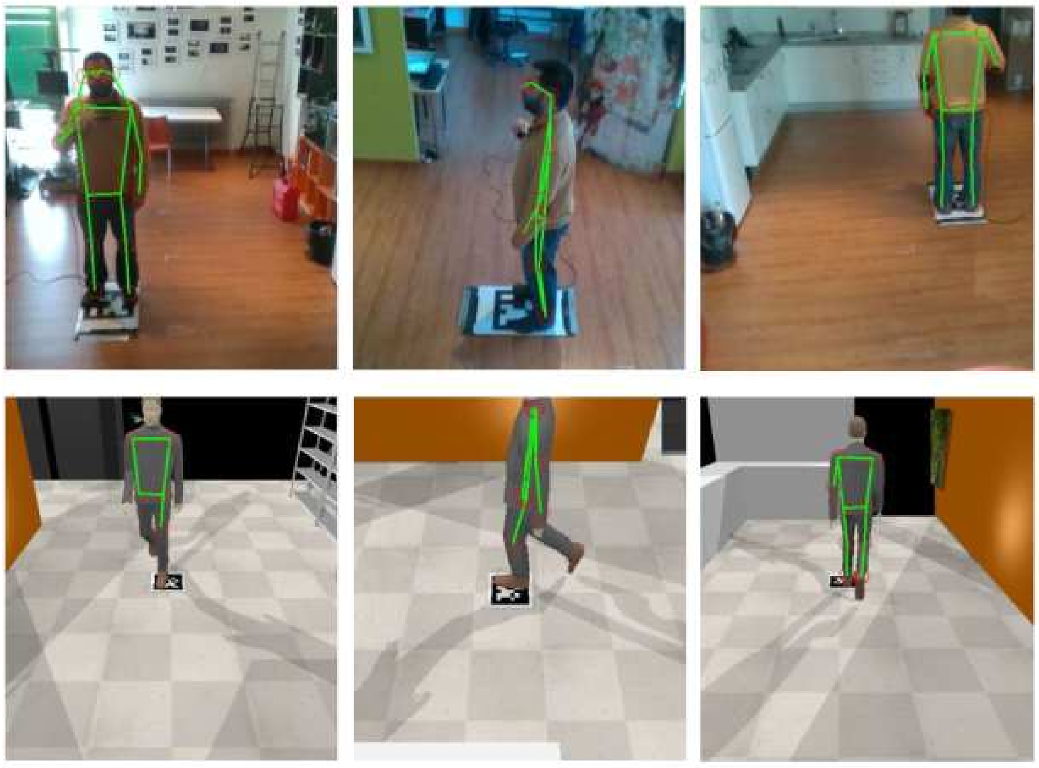


Figure 6.1: Example of real images (first row) and images obtained from the simulator (second row). The resolution employed in the experiments was 640x480. Skeleton detection is visualized with green lines, highlighting the keypoints and connections between them for both real and simulated images.

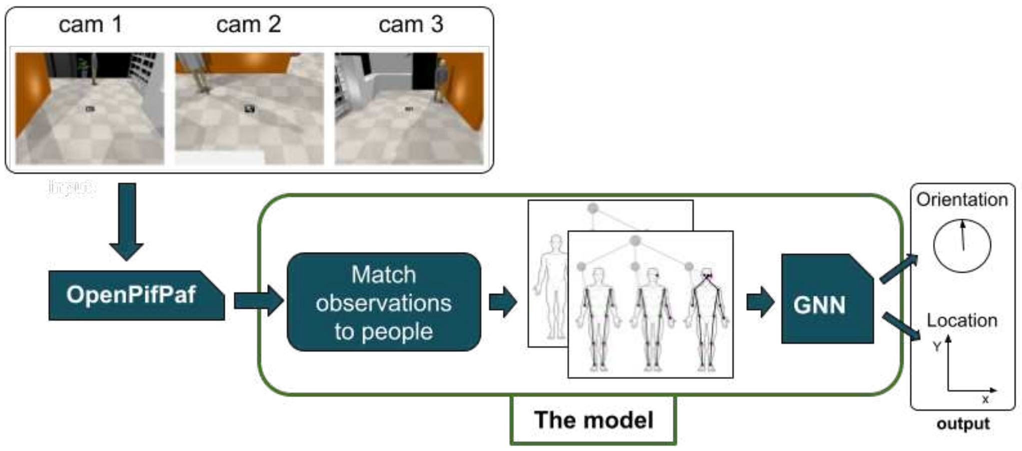


Figure 6.2: Schematic diagram displaying the workflow from image capture via cameras to the prediction of position and orientation. The contributions of this project are encapsulated within the green rectangle in the diagram.

utilization of a GNN in this context is due to its adaptability to potential missing parts detected by the 2D detector, owing to the input dimension flexibility inherent in these networks. Moreover, the graph structures following the body's anatomical form introduce a degree of bias into the network. This bias arguably aids in a better comprehension of the input data, potentially leading to improved results. The remainder of this section explains these phases in more detail.

Image acquisition and skeleton detection: The acquired images are provided to the 2D detector to obtain the skeleton data. Figure 6.1 displays the detected skeletons with green lines for both real and simulated data. For each frame, an observation $\Psi\{p_i, r_i, t_i\}$ is generated and supplied to the next stage, where:

- p_i represents the set of people detected in that frame, each holding a list of up to 16 joint coordinates. If using RGB-D cameras, the depth of each joint from the camera is computed using the depth image plane.
- r_i corresponds to the RGB Region of Interest (ROI) associated with the bounding box of the skeleton.
- t_i indicates the acquisition time of the frame.

Match observations to people: The proposed method for matching observations to people based on geometric cues is not the primary focus of this section; it is merely a suggested approach to solve the problem of cross-view correspondences in this project. However, this method has not been tested and all the experimentation in this section has been done with just one person in the scene. In the following section, a novel solution for this step will be introduced, which aims to improve the matching process and provide better results in the context of multi-camera human pose estimation. Please refer to Section 6.3.1 for a detailed explanation of the new approach and its potential advantages.

A stream of $\Psi_t,\ t \in \mathbb{N}$, observations is generated from the skeleton detectors. A state machine manages the creation, update and removal of a set of data objects representing people. Each observation can either create a new person, update an existing one or be dismissed as noise. Before a new person is accepted, he or she has to receive successful matches for at least 2 seconds.

An observation matches an existing person if their distance $d(o_i, p_j)$ is lower than a certain threshold d_{max}, taken here as 0.65 in the $[0, 1]$ range. The distance d is defined as the median of the distances between the observation and the recent history of the person: $d = med_t \{B(h(o_i), h_t(p_j))\}$ where B is the Bhattacharyya distance [86], $h(o_i)$ is the observation's 2D histogram computed over the hue and saturation planes of the person's ROI and $h_t(p_j)$ is the 2D histogram of person p_j at time t, where t goes from the last observation to Q samples in the past. Other distances and thresholds have been tested with no better results. The removal of unseen people occurs after 2 seconds without receiving any matches.

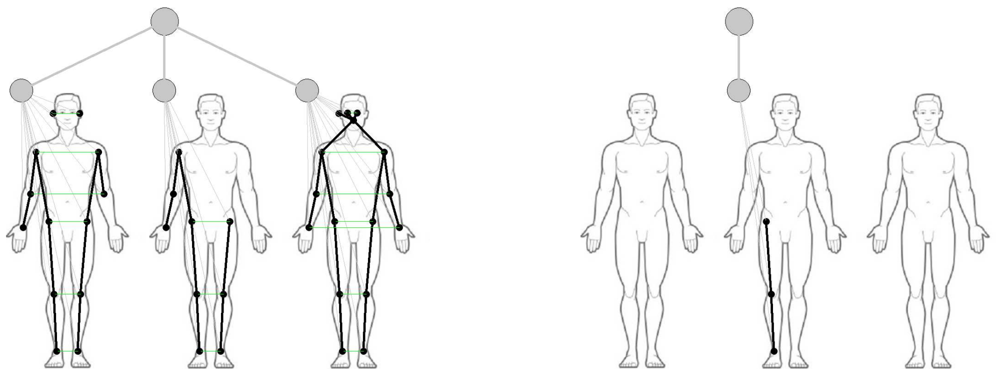
In the next stage, a set S of observations from a person is fed to the GNN to obtain a tuple of target coordinates (x, y, α) representing the pose of each torso. This set S is extracted from the person's history as:

$$S \coloneqq \{o \mid o_i^k \in O^p \wedge k \in \{1, 2, 3\}\}$$

where o_i^k is the past matched observation i by camera k and O^p is the set of past time-ordered observations of person p. S is thus the set of the most recent matched observations obtained from different cameras.

GNN processing: The GNN is designed to use the information obtained from each camera to estimate the position and orientation of a human torso, even with a partial view, yielding more accurate results when more data is available. This stage is the main contribution of the work described in this section. During this phase, two GNN models were tested to tackle the challenge. Specifically, a Graph Attention Network (GAT) and a Relational Graph Convolutional Network (RGCN) were used (see Chapter 3 for more details about these variants). As can be observed in Fig. 6.3, the total number of visible joints is limited in some cases, which makes it difficult to estimate the position and orientation of the person using analytical methods. Instead, as the results suggest, the GNN models effectively leverage the available data to generate more reliable estimates of the human pose, taking into account the partial views from different cameras.

GNNs are particularly well-suited to handling structured data of varying sizes and miss-



(a) Person detected by the three cameras. **(b)** Person partially detected by a single camera.

Figure 6.3: Examples of graph representation of the information obtained by the multi-camera system.

ing nodes (body parts in this case), as discussed in Chapter 3. In the current work, an input graph is created for each set of skeletons associated with the same person. To do this, first, the joints detected by each camera are used to generate separate graphs corresponding to different views of the same skeleton. These graphs contain a node representing the body and additional nodes for all the available body parts (as provided by the 2D detector). Each part is connected not only to its kinematic parent but also to its mirrored body part. The nodes representing the body are referred to as *body nodes*. Finally, all available body nodes (one per view) are connected to an additional node that aggregates information from the previous nodes, which we'll call the *superbody* node. Fig. 6.3 illustrates two graphs with the body parts captured by the three-camera setup.

The feature vectors of the nodes have 25 dimensions (28 if using RGB-D cameras). The feature vector h for each node i is built by concatenating one-hot encodings and metric information:

$$h_i = (t_i|c_i|p_i|s_i)$$

- Node type one-hot encoding (t_i): This encoding represents the node type and has a length of 19 because OpenPifPaf can detect 17 body parts and we use two additional types for the *body* and *superbody* nodes. If another 2D detector is used, the length of this vector would need to be adjusted accordingly.

- Camera one-hot encoding (c_i): This one-hot encoding represents the camera that captures the skeleton. In our experiments, it has a length of 3, as that is the number of cameras used. It is zero-filled for the *superbody* node. To accommodate a different number of cameras, the length of this vector must be adjusted accordingly.

- Coordinates vector (p_i): It has a length of 2 (5 if using RGB-D cameras, adding (x, y, z)) and stores the coordinates of each body part. *Body* and *superbody* nodes have zeros in all the elements of this vector. The image coordinates provided by OpenPifPaf are normalised so that they are within the range $[-1, 1]$. If using $W \times H$ cameras resolution, the normalisation would be:

$$p'_x = (x - W/2)/(W/2)$$

$$p'_y = (H/2 - y)/(H/2)$$

  The 3D coordinates are only provided if using RGB-D cameras. They are also normalised to be in the range $[-1, 1]$, based on the size of the room.

- Score (s_i): This is a single element field that provides the certainty of the measure gathered by OpenPifPaf. It is only used for body part nodes, zero otherwise.

The model is trained so that the output feature vectors are 4-dimensional and correspond to x, y, $sin(\alpha)$ and $cos(\alpha)$ for the *superbody* node in the last layer. The actual angle α is then reconstructed from its sine and cosine.

6.2.1 Dataset Generation

As the models are scenario-specific, they need to be trained with data that incorporate the camera calibration information. Datasets can be created using any simulator that can animate human avatars, provide their ground-truth positions, and offer RGB(D) streams, allowing OpenPifPaf or similar software to detect people and their skeletons. To create an accurate virtual replica, the intrinsic and extrinsic parameters must be estimated (see Appendix C). The accompanying software uses an augmented reality tag placed on the floor (so that it can be detected by multiple cameras) and guides users through the calibration process (see Figure 6.1).

Once the cameras are calibrated, new datasets can be easily produced by generating paths for the simulated avatars in the virtual model. Using this procedure, a large amount of data can be gathered with limited effort. However, simulated data might be insufficient depending on the calibration accuracy, as small calibration errors can lead to a significant reduction in estimation accuracy. For this reason, the dataset generated from the simulations can be extended with real data, recording an actual human moving around the environment. In the experiments, to store ground-truth information, the human was equipped with an Intel RealSense tracking camera on their chest, which provides under 1% closed loop drift. The camera pose at the start coincides with the global reference frame of the room (on top of the AprilTag). Thus, the camera pose directly corresponds to the ground truth pose. Combining both, simulated and real data, a final training dataset with more than 20,000 samples was created. Specifically, 19833 samples of the dataset are simulated and 631 are real. The final dataset is provided in JSON format (available in the public repository[2]).

Each item in the sequence is itself a sequence, having one skeleton for each of the cameras with which a skeleton was detected. In our setup, the number of skeletons for each measurement can be 1, 2 or 3, depending on the number of cameras detecting each person. Each skeleton in the dataset has a ground truth vector (containing the x, z, and yaw of the person), a camera identifier and a timestamp. For each joint the dataset provides the 3D coordinates, the image coordinates of the joint and a certainty value provided by OpenPifPaf). The 3D coordinates of the joints are ignored for the RGB-only experiments.

6.2.2 Experimental Results

The experimental setup consists of three Intel RealSense 415 depth cameras, with extrinsic parameters calibrated concerning a common reference system on the floor (RGB-D cameras are used to enable result comparison using RGB and RGB-D images). Further information about the cameras can be found in Appendix A, and details about the calibration process are available in Appendix C. Once the model is trained, it can estimate 3D poses using only joints' image coordinates, without requiring depth data - although it can optionally be used to enhance results. You can find the code for this project in a public repository[3].

The accuracy of the proposed multi-camera human torso pose estimation system was evaluated through several experiments. Three different sequences of ANN layers were em-

[2]https://github.com/vangiel/WheresTheFellow
[3]https://github.com/vangiel/WheresTheFellow

ployed in each experiment: 1) GAT, 2) RGCN, and 3) per-camera Fully Connected layers (FC), shared across cameras, followed by concatenation and a further sequence of FC layers (MLP). The MLP architecture is provided with 0s and 1s values alongside the normal input to indicate missing data.

All three architectures were trained using the dataset described in section 6.2.1 (**DS1**). Each architecture underwent multiple training sessions with different combinations of hyperparameters applied in each session to identify the most optimal ones (random search hyperparameter tuning). Moreover, a second training dataset (**DS2**) consisting solely of simulated data was generated. The three architectures were also trained using this second dataset, following the same hyperparameter tuning process. For GNNs, various values for the number of layers, number of hidden units, attention heads (for GAT only), number of bases (for RGCN only), and activation function of each layer were tested. These values were randomly generated to cover a wide range of combinations. Likewise, for the MLP, different depths and widths of the hidden layers were explored.

In addition to the two training datasets, two more datasets were generated from real data: one for development, consisting of 225 samples, and another with 283 samples for testing purposes. The collection of larger datasets was not possible due to the COVID-19 pandemic.

Since RGB-D cameras were available, both 2D image coordinates and 3D positions were available for each perceived joint. However, to make RGB-D cameras optional, each architecture was trained using two versions of the data (with and without 3D information). The first version includes the 3D and image coordinates of the body parts in the feature vectors, while the second version only considers the image coordinates, making it suitable for multi-camera systems composed of RGB cameras.

Table 6.4 presents the Mean Squared Error (MSE) of the test dataset for the best model of each architecture, using the 2D-only and 3D versions of the data and the two training datasets. It is evident that the two GNN architectures outperform the MLP in all combinations of training datasets and feature types except for one case in Table 6.3.

As anticipated, using a training dataset with only simulated data (**DS2**) leads to a decrease in accuracy in all cases, with a more pronounced effect on the MLP. This becomes more evident with the 2D version of the data, impacting both orientation (see orientation MSE in table 6.2) and position estimation (see position MSE in table 6.3). However, when using the training dataset that combines simulated and real data (**DS1**), relying on 2D-only features does not significantly impact the results.

	MLP	**RGCN**	**GAT**
DS1 - 3D features	0.024	0.018	0.019
DS1 - 2D features	0.026	0.02	0.021
DS2 - 3D features	0.083	0.080	0.038
DS2 - 2D features	0.077	0.058	0.066

Table 6.2: Orientation MSE for the two training datasets

	MLP	**RGCN**	**GAT**
DS1 - 3D features	0.0011	0.00079	0.00092
DS1 - 2D features	0.0012	0.0016	0.0011
DS2 - 3D features	0.0057	0.0021	0.0013
DS2 - 2D features	0.010	0.0064	0.0049

Table 6.3: Position MSE for the two training datasets

	MLP	**RGCN**	**GAT**
DS1 - 3D features	0.010	0.0076	0.0083
DS1 - 2D features	0.011	0.009	0.009
DS2 - 3D features	0.037	0.033	0.016
DS2 - 2D features	0.037	0.027	0.029

Table 6.4: Global MSE for the two training datasets

To test the accuracy of the solutions, the output was compared with an analytical estimation of the human pose based on the depth data. This comparison enabled us to determine whether the application of deep learning techniques to this problem yielded a more effective solution compared to a purely analytical approach. The position and orientation of the human in the analytical estimation were computed as follows:

- Estimated position: For each camera, the position is individually estimated as the median of the 3D positions of the joints, with the median being used to mitigate the impact of any outliers. The final position is computed by averaging the estimations of all the cameras perceiving at least three joints of the human.

- Estimated orientation: For each camera, the orientation is computed from the positions of pairs of symmetric joints. Specifically, the shoulders and hips are used. The final orientation is obtained as the average of the estimations of the different cameras. If none of the cameras perceives a symmetric pair of joints, the estimation of the previous instant is maintained.

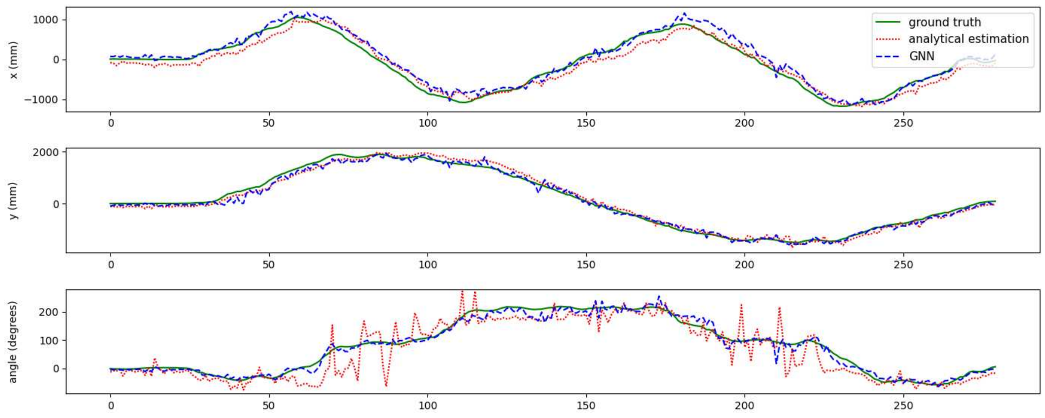


Figure 6.4: Comparison of the network prediction and the analytical estimation of the pose with the ground-truth for every sample of the test datasets.

The comparison between the results obtained from the analytical and the learned estimators shows that learning-based solutions outperform the analytical method, particularly when they have access to the depth channel of the images, especially for orientation estimation. This can be observed in Figure 6.4, which depicts the estimation of the position and orientation for every sample of the test dataset using the analytical method (red dotted line) and the RGCN-based architecture trained using 3D data with the dataset **DS1** (blue dashed line). As can be seen, the difference with the ground truth (green solid line) is smaller in the estimation obtained by the GNN, which is especially remarkable in angle prediction. This observation can be extended to the remaining architectures trained with **DS1** (Figure 6.5). In fact, the mean absolute error (MAE) of the best GNN architecture considerably outperforms the analytical method, providing a **mean absolute angle error below 10°**.

The exclusive use of simulated data for training leads to a deterioration of the results, which can be seen in Figure 6.6. However, the use of 3D information in the data (3D version of **DS2**) still outperforms the analytical estimation for one of the GNN architectures in position and orientation.

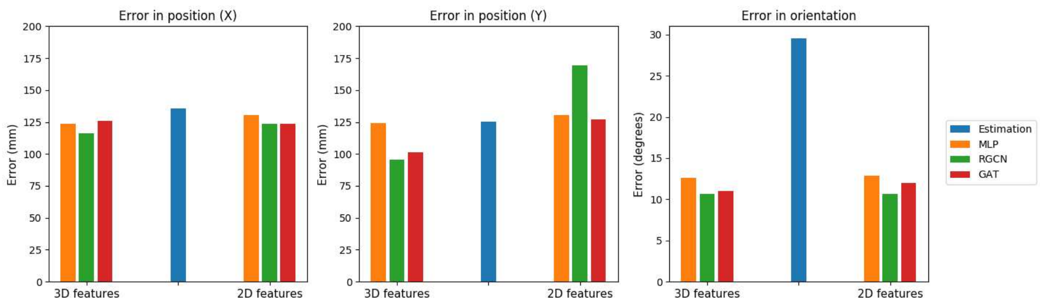


Figure 6.5: MAE of the position (x and y) and orientation for the test datasets using the training dataset DS1, composed of simulated and real data.

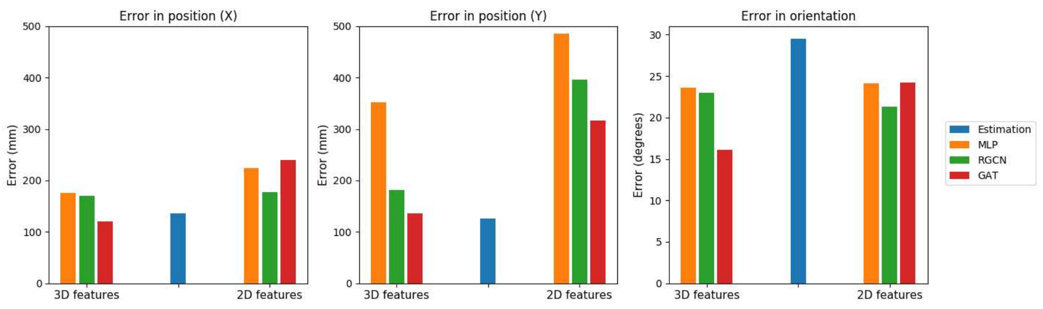


Figure 6.6: MAE of the position (x and y) and orientation for the test datasets using the training dataset DS2 composed entirely of simulated data.

6.3 Multi-Person and Multi-Camera 3D Pose Estimation from Unlabelled Data

This section details the development of a multi-camera and multi-person HPE system capable of predicting 3D positions for principal anatomical keypoints of the human body using RGB cameras. The project presents novel solutions that address some limitations of the system discussed in Section 6.2. Beyond detecting complete skeletons, this advanced system can be trained with unlabelled data through a self-supervised approach, circumventing the necessity for costly equipment and the laborious task of data labelling. Moreover, it introduces a learning-based solution for the pipeline's second step, employing a GNN and achieving exceptional results as outlined in Section 6.3.2. The proposed method is comprehensively described in Section 6.3.1, while Section 6.3.2 presents experimental results comparing the performance of state-of-the-art methods on a public dataset.

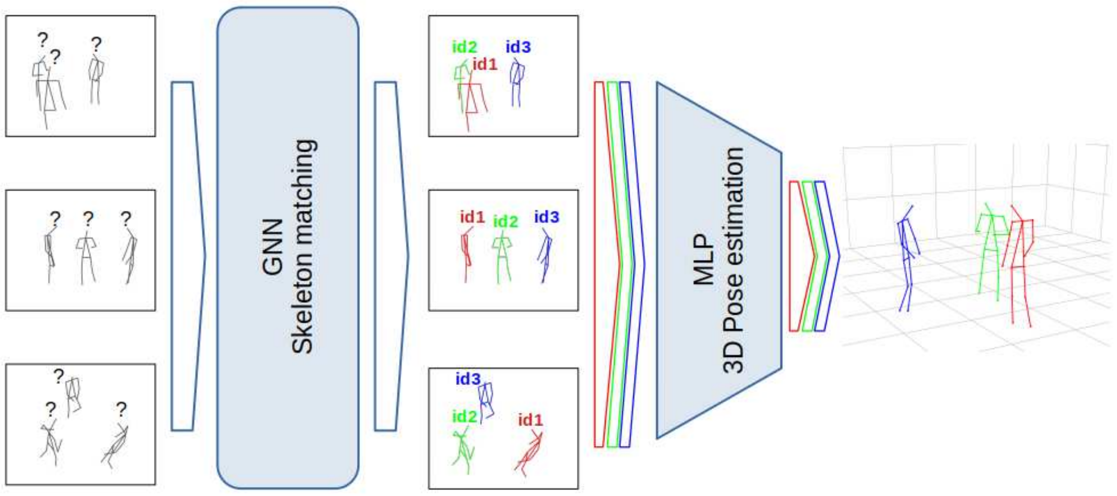


Figure 6.7: Two last stages of the pipeline of the proposed system. The correspondences between the input skeletons in the different views are estimated by a GNN. This information is leveraged by an MLP to provide the final 3D poses.

6.3.1 Method

As already established, multi-view and multi-person systems usually comprise a three-stage pipeline: a) a skeleton detector, b) a multi-view skeleton matching model, and c) a pose estimation model. Given the highly efficient solutions available for the pipeline's first stage, this work does not propose any new alternatives. Indeed, this system is not dependent on the specific skeleton detector employed. The multi-view skeleton matching and the pose estimation network constitute the primary contributions of this section. Figure 6.7 illustrates these two pipeline stages at inference time, which take a set of detected skeletons per view as input that can be obtained using any skeleton detector. The code is accessible at `https://github.com/gnns4hri/3D_multi_pose_estimator`.

System Calibration

The approach presented in this section is not limited to a specific number or arrangement of cameras. However, it necessitates a consistent camera configuration during both dataset collection and final inference for pose estimation. The sole exception is that the system permits the set of cameras utilised during inference time ($\mathbb{C}_i$) to be a subset of those employed during training ($\mathbb{C}_t$). In this scenario, only the cameras in $\mathbb{C}_i$ need to maintain the same configuration as during training. If desired, the remaining cameras in $\mathbb{C}_t$ can be removed from the system once training is complete. This flexibility offers the notable benefit of enhancing inference time accuracy when $\mathbb{C}_t$ can be larger than $\mathbb{C}_i$ (*e.g.*, in mobile robots that require a small set of cameras during inference but can utilise additional cameras for training). This facilitates better *imagination* of the 3D positions of keypoints undetected by cameras in $\mathbb{C}_i$ during inference time thanks to a better reconstruction of the labels during training.

The initial step in setting up the system involves calibrating the intrinsic parameters of all available cameras, as well as their extrinsic parameters relative to the desired global frame of reference. Using these parameters, the projection matrices of all cameras ($T^c; \forall c \in \mathbb{C}_t$) are generated. These matrices are employed during the training and inference phases of the two proposed neural networks, as outlined in the following Sections. For further details on the calibration process, refer to Appendix C.

Skeleton Detection

For training purposes, once the system has been calibrated, a dataset tailored to the cameras' configuration ($\mathbb{C}_i$ and $\mathbb{C}_t$) must be collected. The HPE's training operates under the assumption that the training dataset has been generated with only one person present in the environment at a time. This constraint does not apply to inference time, where the number of individuals is not theoretically limited. Detected skeletons are represented as a list of keypoints, defined by their 2D image coordinates, an identifier for each skeleton keypoint, and a certainty value for the detection. Datasets comprise sequences of samples, each containing a list of detected skeletons per camera.

As previously mentioned, our proposal is not restricted to a specific detector, irrespective of the number of keypoints provided for each skeleton. The number of keypoints merely determines the size of the input features for each network, rendering it a simple configuration parameter.

Skeleton Matching

After detecting skeletons in various views, a critical step involves identifying the skeletons corresponding to the same individual. Given that the order of detections is irrelevant and this HPE aims for the number of views and people to be variable, we train a GNN model to estimate the correspondence between all views. GNNs are well-suited to this task as they are order invariant and accommodate a variable number of input nodes.

The GNN model receives as input an undirected graph $G = (V, E)$, where V is the set of nodes, and E is the set of edges. The set V is composed of two different types of nodes that we term *detection* nodes and *match* nodes. These two types of nodes are the elements of V_d and V_m respectively, such that $V = V_d \cup V_m$. Each *detection* node represents a 2D skeleton detected in one of the views used at inference time (i.e. a view $c \in \mathbb{C}_i$), while a *match* node represents a possible match between two different detections. It is worth noting that only the cameras in $\mathbb{C}_i$ are used because we are only interested in these cameras at inference time and including the rest of the cameras in $\mathbb{C}_t$ would not add any valuable information to this end. For each pair of detection nodes $v_i, v_j \in V_d$ such that v_i and v_j belong to different views, there is a corresponding match node $v_k \in V_m$. The edges in E connect the match nodes to their corresponding detection nodes. Thus, for each match node $v_k \in V_m$ linking two detection nodes $v_i, v_j \in V_d$, there are two edges (v_k, v_i) and (v_k, v_j). Therefore, the input graph G can be represented as follows:

$$G = (V_d \cup V_m, (v_k, v_i) \cup (v_k, v_j)) \tag{6.1}$$

where $v_i, v_j \in V_d$ and $v_k \in V_m$ correspond to the match node between detection nodes v_i and v_j.

Each node (*detection* or *match*) contains a feature vector (x) with $N_k \times N_c \times 10 + 2$ elements, where N_k and N_c are the numbers of keypoints and cameras respectively. Two of these elements denote a binary 1-hot encoding indicating if the node is a *detection* or a *match*. In the case of a *match* node, all other dimensions are fixed to zero. In the case of a *detection* node there is a 10-tuple for each camera-keypoint combination, each of which consists of:

- a flag indicating if the keypoint has been detected,
- the pixel coordinates if the keypoint is visible (2 zeros otherwise),
- a value within the range [0, 1] indicating the certainty of the detection of the keypoint

(zero if the keypoint is not visible),

- six elements encoding the 3D line passing through the origin of the camera and the keypoint (image plane coordinates) in the global frame of reference (specified as a 3D point and a 3D direction vector).

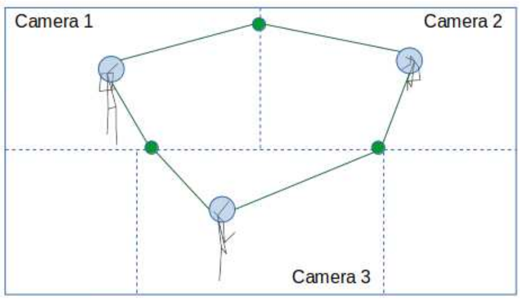
Given the input graph $G = (V, E)$ and the feature vectors of the set of nodes ($x_i \in \mathbb{R}^d$, for $v_i \in V$), a GNN produces an output graph $G' = (V, E)$ with the same structure as G, but different feature vectors for each node ($y_i \in \mathbb{R}^{d'}$). The GNN is trained to predict whether each *match* node $v_k \in V_m$ corresponds to a true match. Thus, the matching is formulated as a binary classification task, where the target labels are $\{0, 1\}$ for non-matches and matches respectively. To ensure that each output of the GNN is within the range of $[0, 1]$, we use the Sigmoid activation function in the output layer. Consequently, both the binary cross-entropy (BCE) loss and the mean squared error (MSE) loss are suitable for computing the loss during the training of the GNN. However, based on experimental results, we observe that the GNN trained with the MSE loss yields slightly better performance than the GNN trained with the BCE loss. Therefore, we define the GNN loss in terms of the MSE loss:

$$\mathcal{L}_{SM} = -\frac{1}{|V_m|} \sum_{v_k \in V_m} (y_k - \hat{y}_k)^2 \tag{6.2}$$

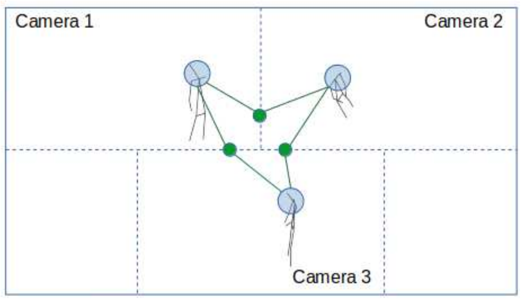
being $y_k \in \{0, 1\}$ the target label for node $v_k \in V_m$, and $\hat{y}_k \in [0, 1]$ the predicted probability that node v_k corresponds to a match.

A primary limitation of learning-based approaches employing supervised learning lies in dataset generation, which necessitates annotation with ground truth. For this specific problem, if the dataset contained multiple skeletons simultaneously, manual annotation of their cross-view correspondence would be required. To circumvent this laborious process, the raw training dataset comprises a set of sequences with single individuals moving within the environment, one at a time. These sequences can then be merged into a single processed dataset containing ground-truth labels, constructed by aggregating data from multiple individuals.

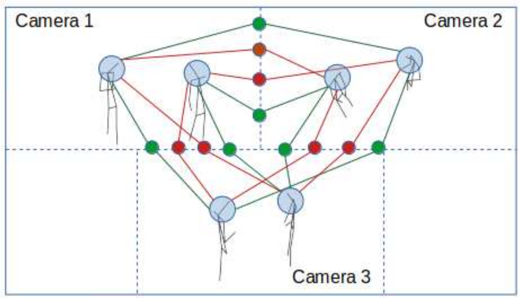
Given that, each frame contains only one person, 2D detections matches can be readily



(a) Individual graph for person 1.



(b) Individual graph for person 2.



(c) Final graph representing two persons.

Figure 6.8: Generation of a sample of the dataset. Graphs of individual persons are generated first assigning a score of 1 to the *match* nodes connecting the views (green nodes). Then a final graph is generated from the individual ones adding *match* nodes with a score of 0 (red nodes).

identified. Using this data, separate graphs are generated for each individual, with all *match* nodes assigned a maximum score value (refer to Figures 6.8a and 6.8b). The graphs of individual persons are then combined by adding *match* nodes with a score of 0, connecting pairs of detections from different individuals, as illustrated in Figure 6.8c. By following this procedure, the target label y_k of each *match* node $v_k \in V_m$ in the graphs composing the training set is generated, enabling GNN training in a pseudo-supervised manner. The number of individual graphs combined is randomly sampled from a uniform distribution for each element in the dataset, with a minimum of one and a maximum equal to the total number of sequences employed in dataset generation.

3D Pose Estimation

Having identified the different views of each person, an MLP is employed to estimate the 3D coordinates of the keypoints for each person. The input features of the model consist of the concatenation of 14 features per keypoint and camera. Consequently, if the skeleton detector identifies up to 25 keypoints and the system utilises three cameras during inference time ($|\mathbb{C}_i| = 3$), the input feature vector dimension would be $14 \times 3 \times 25$, amounting to a total of 1050 dimensions. The 14 features per keypoint include the 10 features delineated in the previous section for skeleton matching, along with four additional features related to an initial 3D estimation. Specifically, if a keypoint of an individual is detected by two or more cameras, its 3D coordinates are reconstructed via triangulation for each pair of cameras, and an initial estimation is computed as the centroid of the obtained 3D points. This estimation is incorporated as input using three of the four new features. The final feature is employed to indicate the availability of the estimated 3D. It is set to 1 if there is more than one view of the keypoint and 0 otherwise.

Utilising the aforementioned information per keypoint and camera, the network estimates the 3D coordinates of all keypoints in the global frame of reference. Thus, assuming that the network predicts the position of 25 distinct keypoints, the output vector dimension is $3 \times 25 = 75$. An MLP query is performed for each person.

The training process follows a self-supervised learning approach, which represents the main advantage of this approach. This way, there is no need to use a ground truth to compare the output, since the loss function only uses the data from the skeleton detectors. However, calculating this loss is not trivial, since the network infers 3D poses from 2D image coordinates. The approach to solving this problem is to project the 3D coordinates of the keypoints predicted by the network into each camera used for training ($\mathbb{C}_t$). The transformation between global and image coordinates is done by using the projection matrices ($T^c\ \forall c \in \mathbb{C}_t$) obtained during the calibration process. Using the projected coordinates and the coordinates yielded by the skeleton detector, a measurement of the estimation error of the network is obtained. This error defines the loss function that the network is trained to minimise. More formally, assuming that the output of the network o is represented as a vector of 3D positions corresponding to the estimation of the person's keypoints coordinates:

$$o := (o_0, o_1, ..., o_{N_k-1}) \tag{6.3}$$

with N_k the number of keypoints, a vector p^c of image projected positions (p_i^c) can be obtained for each camera as follows[4]:

[4]The conversions between homogeneous and standard coordinates are omitted for simplification

$$p_i^c = T^c \cdot o_i \quad \forall i \in [0, N_k) \tag{6.4}$$

Using p^c and the set of detected keypoints ($S^c = \{s_k^c\}$) for each camera c, the projection error e is computed as

$$e = \sum_{c \in \mathbb{C}_t} \sum_{s_k^c \in S^c} d(p_k^c, s_k^c) \tag{6.5}$$

being $d(\cdot)$ the Manhattan distance between the projected and detected points.

Applying equation 6.5 to each sample of the dataset D, the final loss is calculated using the mean squared error:

$$\mathcal{L}_{3D} = \frac{1}{|D|} \sum_{d \in D} e_d^2 \tag{6.6}$$

being e_d the result of equation 6.5 for the sample d.

Figure 6.9 depicts the process to compute the self-supervised loss, assuming 25 keypoints, 4 cameras in $\mathbb{C}_t$, and 3 cameras in $\mathbb{C}_i$. It can be observed that the loss computation utilises detections from all cameras in $\mathbb{C}_t$ but only the cameras in $\mathbb{C}_i$ are used as network's input. Consequently, in the aforementioned example, the model receives detections from only three of the four cameras at inference time. However, even though the fourth camera would not be used at inference time, our HPE would still exploit what was learned from it at training time.

Data augmentation

Data augmentation is applied to extend the data used for training, to increase the variety of situations, and to increase the robustness against partial views. Specifically, for each original sample of the dataset, which we refer to as *seed samples*, several samples are generated removing views. Given a sample s comprising data obtained from n different views ($s = d_1, d_2, ..., d_n$), m new samples can be generated by selecting subsets of the views. The subsets

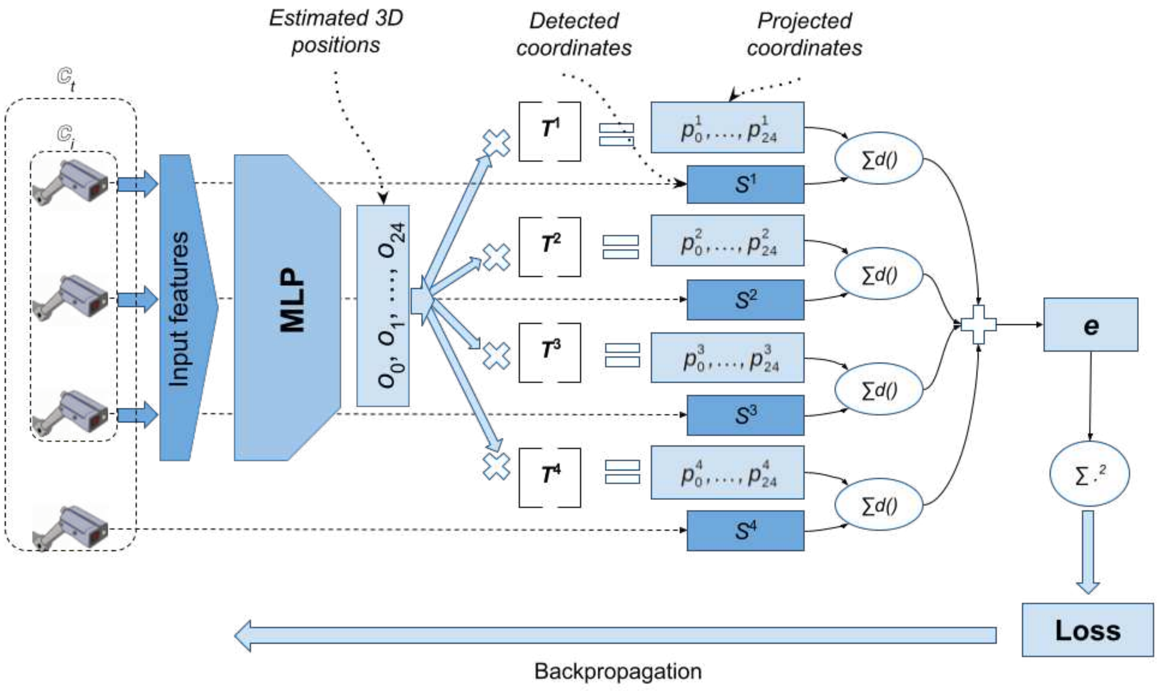


Figure 6.9: Representation of how the 3D pose estimation network training loss is computed in a setup with 4 cameras in $\mathbb{C}_t$ and 3 cameras in $\mathbb{C}_i$.

are chosen randomly from a uniform distribution in the range of all possible combinations that can be obtained with the different number of views (from 1 to n). For example, suppose a seed sample s contains data from 5 views ($s = \{d_1, d_2, d_3, d_4, d_5\}$), and we want to generate 3 new samples from s, the following samples could be randomly selected from the possible view combinations and added to the dataset: $\{d_1, d_3, d_4\}$, $\{d_2, d_5\}$, $\{d_3\}$.

The new data generated by this process are used as the input of the two networks when training. However, in the case of the pose estimation network, for each generated sample, the whole data of its seed sample is used in the computation of the loss (equation 6.6), as losing self-supervision information would not be beneficial.

6.3.2 Experimental results

The 3D multi-human pose estimation system has been evaluated using the CMU Panoptic Studio dataset [84] and a dataset generated at Aston University's Autonomous Robotics and

Perception Laboratory (further information about this laboratory is available in Appendix A) for the purpose of this research. The experiments presented in this section offer empirical evidence that the proposed approach outperforms other state-of-the-art algorithms in terms of speed, maintains comparable accuracy, and crucially, does not necessitate annotated datasets. Lastly, evidence is provided to demonstrate that the proposed HPE can effectively operate on an autonomous robot.

To conduct these experiments, the two neural models were trained for each dataset using a train/validation/test split to avoid data leakage. For the matching model, a Graph Attention Network (GAT) is employed (see Chapter 3), with four hidden layers. The hidden layers comprise $[40, 40, 40, 30]$ hidden units and $[10, 10, 8, 5]$ attention heads. LeakyReLU and Sigmoid serve as activation functions for the hidden and output layers, respectively. The MLP-based pose estimator features seven hidden layers with $[3072, 3072, 2048, 2048, 1024, 1024, 1024]$ hidden units, using LeakyReLU for the activation of the hidden layers and linear activation in the output layer.

Datasets

To evaluate the proposed approach on the **CMU Panoptic dataset** and enable comparisons, the same sequences and views as VoxelPose [177] are used. Likewise, the four sequences employed for testing VoxelPose are applied in these experiments. The data containing the 2D skeletons' information were obtained using the backbone model provided in the VoxelPose project. This model allowed for the detection of the 2D coordinates of humans' keypoints from the images. However, the training strategy necessitates that each data sample includes the information of only one person, so it was essential to arrange the detection results to provide individual human data. To achieve this, the skeletons of different views corresponding to the same human were identified using the ground truth of the Panoptic sequences and grouped to obtain individual samples for each human.

As mentioned in section 6.3.1, the proposed approach assumes a fixed configuration of cameras for both the training and inference phases. All the cameras must be calibrated according to a global frame of reference before training. However, this requirement is not strictly met in the Panoptic dataset, which results in some limitations when compared with the ground truth. To illustrate the problem, table 6.5 displays the translation vector to the global frame of reference for the five selected cameras in two of the datasets. As can be seen, there are significant variations between the positions of the cameras in the two datasets, exceeding, in some cases, and for specific axes, 0.1m. This implies that each sequence considers a different global frame of reference.

To address this limitation, the calibration file of one of the datasets (160224_haggling1) is used for training, and, for testing, the ground truth of each test dataset is transformed from the global reference frame of that dataset to the global reference frame used for training. To apply this transformation, a specific camera serves as a common frame of reference for all the datasets. Consequently, the ground truth is first transformed from the global frame of reference of the dataset to the camera frame of reference and then from the camera frame of reference to the global frame of reference used for training. Although this transformation partially resolves the problem of having different calibration data for each dataset, a residual error remains due to minor variations in the intrinsic and inter-camera extrinsic parameters in the Panoptic sequences. Despite this drawback, which impacts negatively only our approach, the results are still comparable.

	Sequence					
	160224_haggling1			**160422_haggling1**		
Camera	X	Y	Z	X	Y	Z
HD03	2087.71	-1510.89	1780.99	2015.19	-1512.49	1789.8
HD06	-677.9	-3394.66	-1704.22	-641.81	-3398.11	-1783.39
HD12	-76.23	-2392.45	2552.27	-173.19	-2395.36	2494.95
HD13	-1840.95	-3393.29	143.76	-1860.28	-3393.9	28.87
HD23	2343.16	-1526.22	-1433.97	2372.17	-1527.83	-1417.28

Table 6.5: Translation (in millimeters) between each camera and the global frame of reference for two sequences of the CMU Panoptic dataset.

The **ARP Laboratory dataset** was generated from 4 cameras attached to the walls of the laboratory and 2 additional cameras mounted on a mobile robot. The robot was static and located at a fixed position during the generation of the dataset. All the cameras were calibrated in relation to a global frame of reference. A total of 18 video sequences of single individuals moving were recorded. The sequences have variable lengths between $2'$ and $39'$. These sequences were used for training and testing separately the two models. Two additional sequences with groups of 2 and 4 people were recorded to test the whole system. These test sequences have a length of $3,43'$ and $2,58'$, respectively.

Evaluation of the Skeleton-Matching Module

Since the goal of the skeleton-matching network is to group together the different views of a person, given an unknown number of people, it can be considered a clustering model. Therefore, the proposed matching technique can be evaluated using a set of clustering metrics. Specifically, the following metrics have been used:

- **Adjusted rand index** (ARI) [78]: estimates the similarity between two clusterings according to the number of pairs belonging to the same or different clusters. It is adjusted using a random model as a baseline, ensuring a random clustering has a value close to 0. This score ranges between -0.5 (discordant clustering) and 1.0 (perfect clustering).

- **Homogeneity** (H) [152]: measures the homogeneity of the clusters. A cluster is considered homogeneous if it contains only members of the same class. It ranges between 0.0 and 1.0.

- **Completeness** (C) [152]: measures the completeness of the clusters. A cluster is considered complete if all the members of the same class are assigned to the same cluster. It ranges between 0.0 and 1.0.

- **V measure** (Vm) [152]: harmonic mean between homogeneity and completeness. This index quantifies the goodness of the clustering, considering both homogeneity and completeness. It ranges between 0.0 and 1.0.

These metrics have been applied to several skeleton matching networks trained for different numbers of views using the two datasets described in the previous section. Table 6.6 shows the results for two, three, and five views using the four test sequences of the CMU Panoptic dataset. For all the metrics, values close to 1 are obtained regardless of the number of views, which means almost perfect scores in each of these metrics.

The effectiveness of the proposed skeleton-matching network was also evaluated using the ARP Laboratory dataset. Two models, one with two views and the other with six views,

No. of views	ARI	H	C	Vm
2	0.9875	0.9968	0.9925	0.9943
3	0.9977	0.9993	0.9981	0.9986
5	0.9941	0.9978	0.9937	0.9956

Table 6.6: Metrics of the skeleton matching network for the CMU Panoptic dataset.

were trained using ten of the eighteen sequences of single individuals. The models were then tested on the remaining eight sequences, with a test dataset generated according to the multi-person dataset generation process detailed in section 6.3.1. This process provided the necessary ground truth to compute the evaluation metrics.

Table 6.7 presents the results obtained from 2000 samples in the generated dataset, which contained varying numbers of persons ranging from 1 to 8. Similar to the CMU Panoptic dataset, the evaluation metrics demonstrated outstanding performance of the network for both the two and six views models. Furthermore, it is noteworthy that for both datasets, the homogeneity values are nearly 1, indicating that the skeleton groups are predominantly comprised of views from the same individual.

No. of views	ARI	H	C	Vm
2	0.9770	0.9966	0.9886	0.9923
6	0.9842	0.9974	0.9847	0.9905

Table 6.7: Metrics of the skeleton matching network for the ARP Laboratory dataset.

Evaluation of the Multi-Person 3D Pose Estimation System

The whole multi-person 3D pose estimation system has been evaluated for both, the CMU Panoptic and the ARP datasets. For CMU Panoptic, the same metrics in [177] have been used for comparative purposes:

- **Mean per joint position error** (MPJPE): mean distance (mm) per keypoint between detected and ground truth poses.
- **Mean average precision** (mAP): mean of average precision over different distance thresholds (from 25mm to 150mm, taking steps of 25mm).
- **Mean recall** (mR): mean of recall over all the thresholds.

- **Time for persons' proposals** (t_{pp}): the mean time required for generating persons' proposals. In our approach, this time corresponds to the skeleton matching stage.
- **Time for 3D pose estimation** (t_{3Dg}): mean time required for estimating the 3D poses.
- **Time for 3D pose estimation per human** (t_{3Di}): mean time required for estimating the 3D pose of one person.

To facilitate a comparison with alternative methodologies, VoxelPose was trained using the same ten Panoptic training sequences. Additionally, the performance of the pose estimation model was compared with 3D poses derived through triangulation. More specifically, for each pair of views corresponding to an individual identified by the skeleton matching model, the 3D position of each visible keypoint was estimated by triangulating the 3D coordinates of its respective 2D counterparts. In instances where multiple estimations were procured (i.e. when a keypoint was visible from more than two cameras), the final 3D position for the keypoint was calculated as the mean of individual estimations.

Regarding the proposed approach, three distinct versions of the test dataset were employed. In the first version (*D-detected*), the input skeletons contained 2D coordinates of the keypoints detected by VoxelPose's backbone skeleton detector. In the second version (*D-projected*), the 2D positions of the detected keypoints were substituted with the projected coordinates of the ground truth 3D keypoints. Lastly, in the third version (*D-average*), the 2D positions of the keypoints were computed as the mean between the detected and projected 2D coordinates.

The main reason for using these 3 versions stems from the significant discrepancy between the detected 2D keypoints and those acquired by projecting the ground truth 3D skeletons. This observation is evident in Table 6.8, which displays the reprojection error of the ground truth for the three datasets. To create this table, the ground truth 3D was projected onto the images of the five cameras, taking into account the calibration data used to train the model, and compared with the 2D keypoints of the three datasets. As observed, substantial differences exist among the reprojection errors when considering the three datasets. As anticipated, the largest reprojection error occurs in the *D-detected* dataset, indicating discrepancies in the human body keypoints' positions between the skeleton detector model and the Panoptic datasets' ground truth. Furthermore, certain disparities are evident between the reprojected ground truth and the keypoint positions in the *D-projected* dataset. These differences are related to the different calibration data of each sequence of Panoptic. Specifically, as mentioned in section 6.3.2, the variations of the intrinsic and inter-camera extrinsic parameters of the sequences produce a remaining error that is reflected in the second row of table 6.8. In fact, compared with the *D-avarage* dataset, cameras *HD03* and *HD23* present higher reprojection errors for the *D-projected* dataset.

	Camera				
Dataset	HD03	HD06	HD12	HD13	HD23
D-detected	7.01	10.73	7.63	10.71	6.37
D-projected	3.81	1.08	2.19	2.28	4.74
D-average	3.65	5.12	3.92	6.06	3.69

Table 6.8: Reprojection error of the ground truth 3D for the three versions of the test dataset using CMU Panoptic.

Table 6.9 presents a summary of the accuracy and time metrics obtained for VoxelPose, triangulation, and the proposed method across the four test sequences of the CMU Panoptic dataset. The final three rows of the table represent the performance of the proposed model on the three dataset variations (*D-detected*, *D-projected,* and *D-average*).

Method	MPJPE	mAP	mR	t_{pp}	t_{3Dg}	t_{3Di}
VoxelPose	17.97	96,61	97,41	135.92	169.99	50.53
Triangulation	22.63	76,99	85,10	32.56	10.06	2.99
Proposed-detected	26.06	89,25	92,63	31.67	19.65	5.83
Proposed-projected	17.84	96,23	97,76	31.96	19.94	5.89
Proposed-average	19.77	95,67	97,39	32.22	19.81	5.85

Table 6.9: Accuracy and time metrics of VoxelPose, triangulation, and proposed method using the CMU Panoptic dataset.

In terms of accuracy, VoxelPose and the proposed model for the *D-projected* dataset exhibit similar performance, despite the fact that the proposed model was trained with detected data. Moreover, the results of the proposed model on the *D-average* dataset for *MPJPE*, *mAP*, and *mR* are quite comparable to those of VoxelPose. The lowest value of *MPJPE* is observed for the proposed model using the *D-detected* variation of the test dataset. As previously mentioned, this is attributable to the discrepancies between the 2D detected and ground truth projected coordinates. Nonetheless, triangulation yields a similar mean position error, even though its computation solely considers keypoints visible from two or more cameras. Furthermore, triangulation performs the worst in terms of mAP and mAR, primarily due to its inability to consistently yield complete pose estimations. Figures 6.10 and 6.11 provide examples of complete and incomplete results using triangulation. Figure 6.10 displays instances where all keypoints for every individual in the scene can be estimated by triangulation. In these cases, the estimated poses provided by the proposed model (images on the left) and triangulation (images on the right) closely resemble the ground truth poses (depicted in grey). Conversely, in Figure 6.11, some poses cannot be fully determined by triangulation, as certain keypoints are not visible from two or more cameras. In such situations, the proposed model offers complete estimates for all poses, with minimal deviations from the ground truth.

Interestingly, the second scenario in Figure 6.11 presents a situation where the ground truth is incomplete (green skeleton). In this case, it is evident that the proposed model provides a realistic pose despite the scarcity of information.

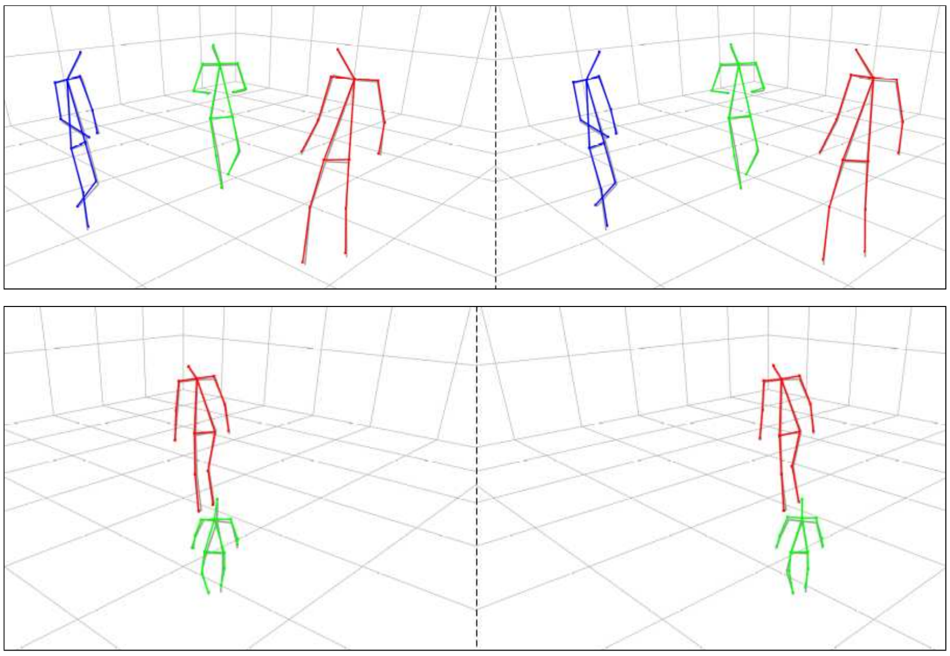


Figure 6.10: Pose estimation results for 2 samples of the test sequences using the proposed model (left images) and triangulation (right images). The ground truth is shown in grey. Triangulation provides complete poses in the 2 samples.

Concerning computational time, VoxelPose requires an average of 305.91 ms for the

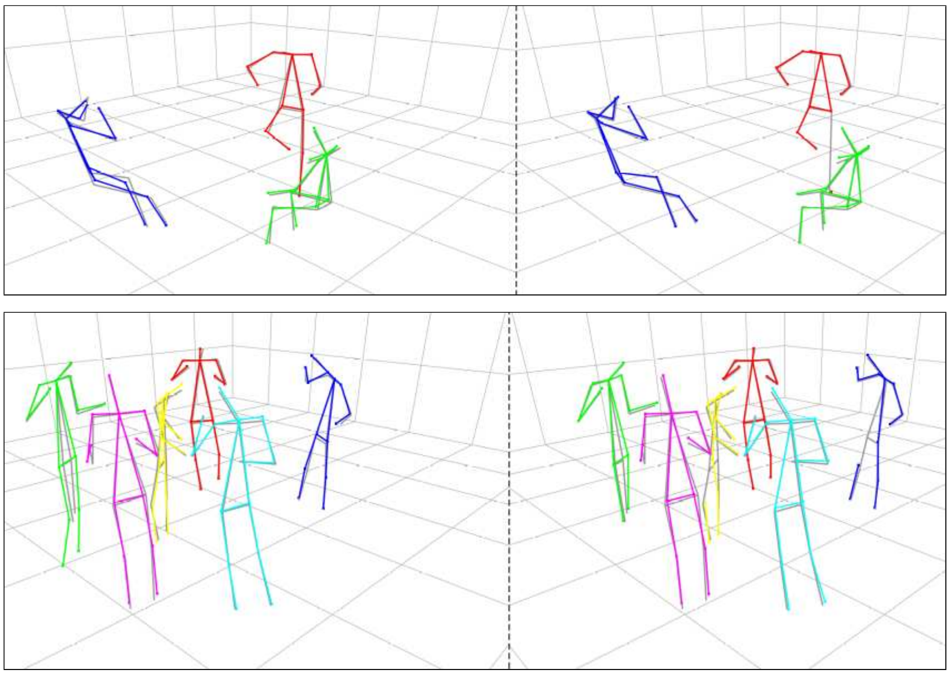


Figure 6.11: Pose estimation results for 2 samples of the test sequences using the proposed model (left image) and triangulation (right image). The ground truth is shown in grey. In these samples, triangulation cannot provide complete poses due to an insufficient number of views for some keypoints.

entire estimation process, which is nearly six times longer than the duration needed by the proposed method. The metrics of table 6.9 do not include the time necessary for skeleton detection, which may vary depending on the specific detector employed. Nevertheless, efficient solutions for detection do exist, such as *trt-pose*, which can perform at 251 FPS on Jetson Xavier [192]. Moreover, skeleton detection for all views can be executed in parallel, rendering the timing largely independent of the number of views. Thus, assuming skeleton detection can be achieved at 30 FPS, the proposed method still operates more than three times faster than VoxelPose.

Besides the aforementioned benefits regarding real-time execution, the proposed approach offers a more versatile solution to the problem compared to existing alternatives. The fact that **no ground truth is required** to train the two models makes the proposed method easily replicable, regardless of the space, organization and extension.

The proposed multi-person 3D pose estimation system has also been evaluated using the ARP Laboratory dataset. As this dataset lacks ground truth, the accuracy metrics employed for the CMU Panoptic dataset cannot be applied. Instead, the reprojection error

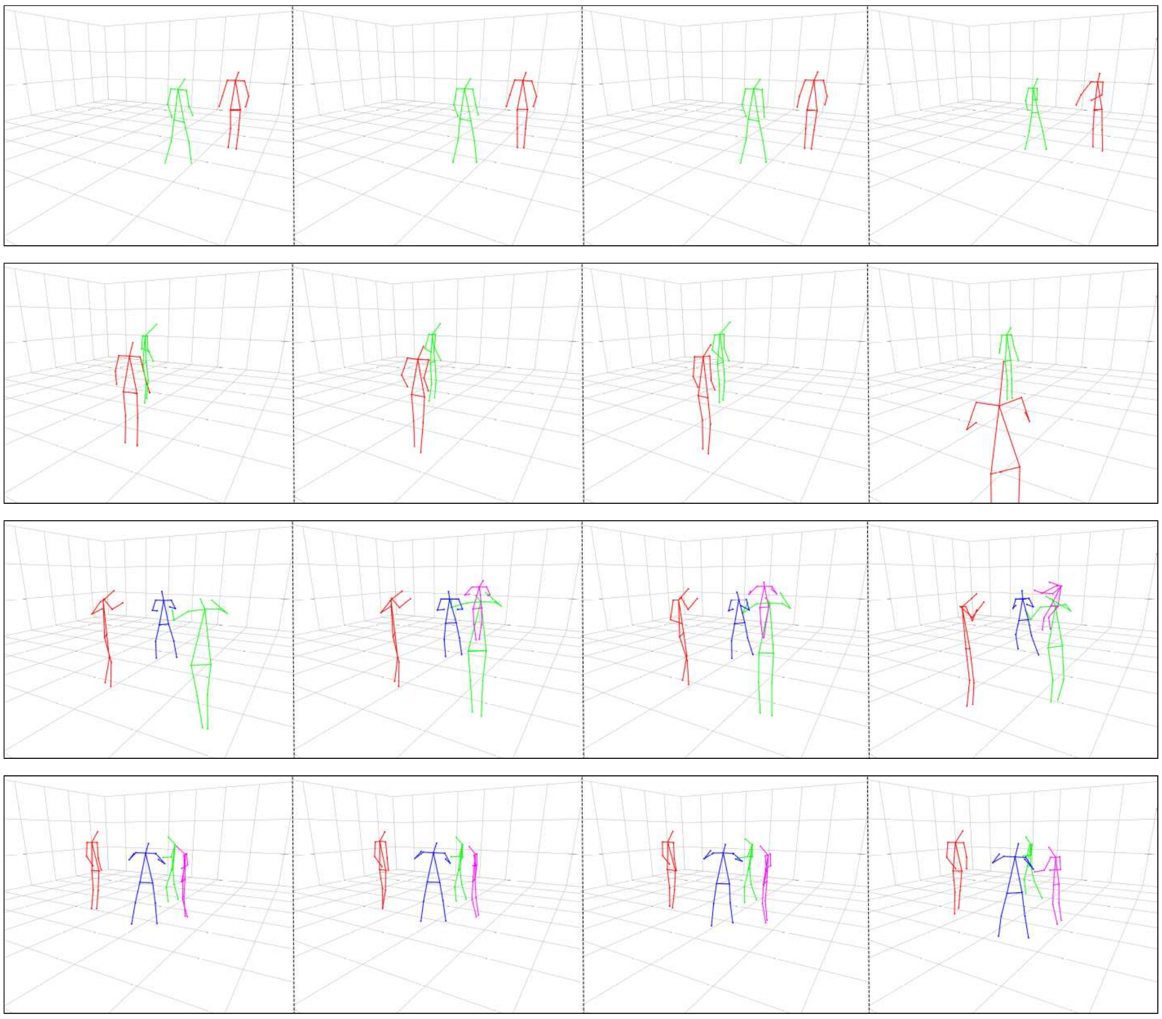


Figure 6.12: Pose estimation results from the proposed proposal of four samples of the ARP Laboratory multi-person sequences. From left to right, the results correspond to the models $M_{1/6}$, $M_{2/6}$, $M_{6/6}$, and $M_{2/2}$.

of the estimated 3D for all cameras is utilised. Four distinct models were trained using different sets of cameras at training and inference times: $M_{6/6}$, which employs the six cameras for both training and inference; $M_{2/6}$, which utilises the six cameras for training and the two robot-mounted cameras for inference; $M_{1/6}$, which is trained with the six cameras but only employs one of the robot's cameras during inference; and $M_{2/2}$, which exclusively uses the two robot-mounted cameras for both training and inference.

Mean and median reprojection error for the four models and the six cameras (wall cameras: W0, W1, W2, and W3; robot cameras: R0 and R1) using the two ARP Laboratory sequences with 2 and 4 people are depicted in table 6.10. The lowest reprojection error for the wall cameras is given by the model $M_{6/6}$. This model is the most reliable since it uses data from all the views. Despite models $M_{2/6}$ and $M_{2/2}$ using the same cameras at inference time, there are significant differences in their behaviour, which is reflected in the lower reprojection errors of $M_{2/6}$. Especially, it can be observed a very high error of $M_{2/2}$ for the camera $W1$. Such a large error is produced when there is limited visibility of a person from the cameras used by the model. This is the case with the second sample of figure 6.12, where the model places the red skeleton far away from its actual position. Besides this specific case, in general, the keypoints positions estimated by model $M_{2/2}$ differ from the estimates of model $M_{6/6}$ as observed in that figure. In contrast, the model $M_{2/6}$ can estimate the pose of the person correctly, even though the input of both models is common. Generally, the poses provided by the model $M_{2/6}$ are very similar to those provided by the model $M_{6/6}$, as demonstrated by both figure 6.12 and the reprojection errors in table 6.10. Finally, the model $M_{1/6}$ shows outstanding results considering it only receives information from one of the cameras of the robot (*R0*). The model is capable of predicting complete 3D poses, producing comparable reprojection errors to model $M_{2/6}$ [5].

The results of this experiment demonstrate that the proposed system is capable of providing good estimations with a reduced number of cameras, by considering the information of an extended set of cameras during training. This is a significant advantage for its application in autonomous robots, which is the focus of the next section.

Camera	**Model**			
	$M_{1/6}$	$M_{2/6}$	$M_{6/6}$	$M_{2/2}$
W0	14.65 / 11.26	14.35 / 11.00	10.28 / 8.23	45.73 / 26.85
W1	11.84 / 9.28	12.02 / 9.49	8.28 / 6.80	290.63 / 15.62
W2	14.02 / 10.68	14.04 / 10.55	11.12 / 8.41	29.78 / 21.86
W3	12.18 / 9.29	12.36 / 9.40	7.88 / 6.40	31.94 / 19.24
R0	6.69 / 5.27	7.06 / 5.34	8.98 / 6.97	4.50 / 3.37
R1	9.05 / 6.83	7.79 / 6.07	9.49 / 7.51	4.50 / 3.38

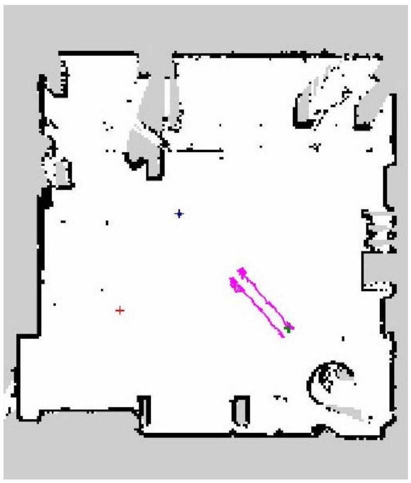
Table 6.10: Mean and median reprojection error in the 6 cameras of the ARP Laboratory for 4 models trained with different numbers of train and inference cameras.

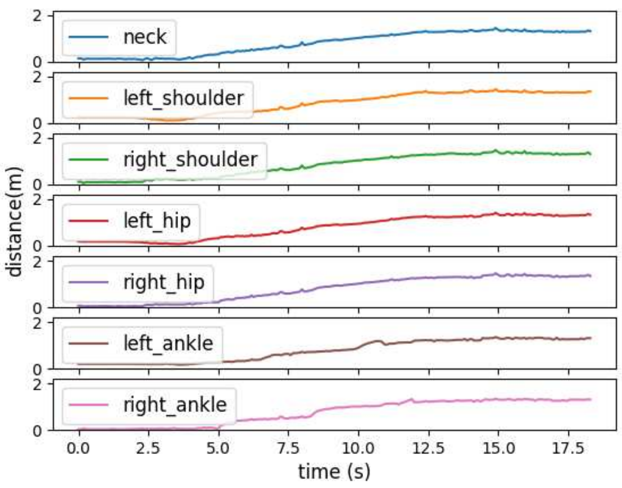
[5]Bear in mind that, even though the reprojection errors are lower for $M_{1/6}$ than for $M_{2/6}$ in four of the six cameras, non-visible people from camera *R0* (see the third sample of figure 6.12) are not considered in the error computation of model $M_{1/6}$.

Evaluation in a Mobile Robot

This experiment aims to show the application of the proposed system in a mobile robot equipped with only two RGB cameras. The main goal is to endow the robot with the ability to estimate the complete 3D human poses with enough accuracy to enhance the interaction between them.

Since the robot does not stay in a fixed location, the only visual information it can use is that provided by its two cameras. In the case of a mobile robot, triangulation is less informative. The short baselines of robots' stereo systems rarely provide complete poses, and more importantly, they can cause small deviations in keypoints' image positions to produce large 3D errors. Nevertheless, using only the data captured by the two cameras to train the pose estimation model does not provide reliable results, as shown in the previous section. For this reason, we use the model $M_{2/6}$, which only requires the information of the two cameras on the robot at inference time, but uses the data from the four additional wall cameras during training.

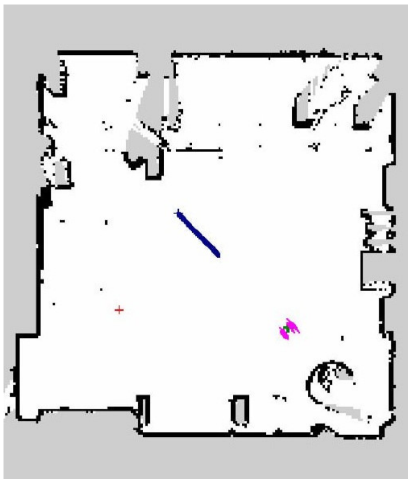

(a) Trajectory of the person's ankles.

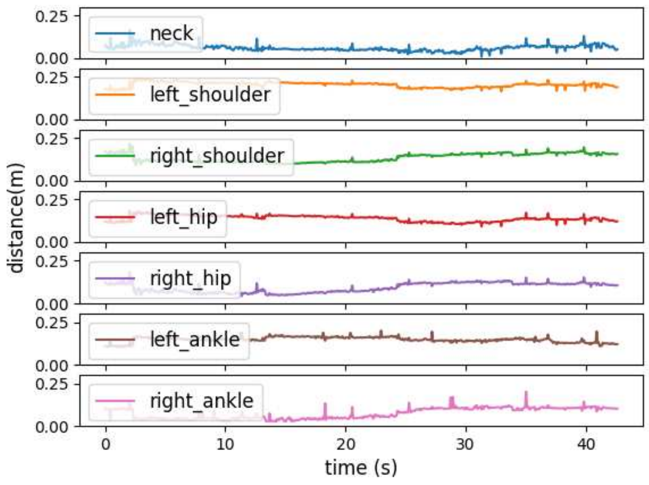


(b) Distance from the initial position of the person to a subset of keypoints.

Figure 6.13: Experiment maintaining the robot in a fixed position while a person walks 1.5 meters towards the robot. Only two RGB cameras are used at inference time. It is expected that the positions go from 0 meters to 1.5 meters in the graphs.

Two distinct experiments were conducted to validate the effectiveness of the proposal. In the first experiment, the robot remains stationary, facing the location of a person situated 2.75 meters from the front part of the robot. The 3D pose of the individual is recorded as they walk towards the robot in a straight line, covering a distance of approximately 1.5 meters. Figure 6.13a illustrates the displacement of the person's two ankles (magenta lines), along with the position of the robot (dark blue cross). The green cross on the map represents the initial position of the person, and the red one denotes the position of the global frame of reference. Figure 6.13b displays the displacement in meters of the coordinates (projected on the floor plane) of some representative keypoints from the initial position. As observed, the travelled distance for all keypoints ranges from near 0 to roughly 1.5 meters. Additionally, the distance between symmetric keypoints exhibits only minor variations along the entire route (*e.g.* the standard deviation is 2.3 cm for the hips and 0.64 cm for the eyes), which indicates the stability of the estimations.


(a) Trajectory of the robot.



(b) Distance from the initial position of the person to a subset of keypoints.

Figure 6.14: Experiment where the person remains still while the robot moves 1 meter towards the person. Only two RGB cameras are used at test time. As the person is not moving the positions should remain the same throughout time.

The second experiment employs the same setup, with the person remaining stationary while the robot approaches them following a straight line covering a distance of 1 meter. Figure 6.14a shows the robot's path (thick blue line), extracted from its localisation system, and the position of the person's ankles, with the initial point situated between them. Figure 6.14b displays the distances from the initial position of the person to the last. At every instant, the position is calculated as the projection onto the floor of the same representative keypoints as in the previous experiment. As shown in this figure, the distances remain almost constant, which is the expected result. As in the previous experiment, the variations in the distances between symmetric keypoints are insignificant, with values from 1.8cm for the ankles to 0.8cm for the hips, which demonstrate the robustness and good accuracy of the model.

6.4 Conclusions

The torso pose estimation system explained in Section 6.2 is specifically designed for spaces monitored by multiple cameras. It is assumed that individuals are visible to at least one camera, although certain body parts may be occluded or beyond the field of view of some cameras.

In comparison to other works, the proposed torso estimation system outperforms [161] and it is -under good conditions- outperformed by [108]. Although [108] reports better results, it requires RGB-D cameras, which are an order of magnitude more expensive than low-resolution RGB cameras. Additionally, as reported in [108] their dataset does not consider occlusions or partial views, so their results will likely deteriorate in real-life conditions.

A mean absolute error of $125mm$ in the torso pose coordinates is deemed acceptable for the majority of human-robot interaction tasks, such as human-aware navigation. This assessment takes into account: **a)** the average human size, *i.e.*, the 50th percentile of adult forearm-forearm breadth measures approximately $492mm$ (female) and $579mm$ (male)[62]; and **b)** proxemics research that identifies personal spaces as approximating a circle of around $1200mm$ in diameter[135].

Given the accuracy attained using standard RGB images and the marginal improvements achieved through depth channel utilisation, conventional low-end webcams are arguably enough for most HRI scenarios. Furthermore, camera calibration may be rendered unnecessary if the dataset is solely derived from real situations. However, employing a realistic simulator to generate the majority of data for training significantly reduces the time and resources required to obtain a valid solution for the pose estimation problem.

Although the torso pose estimator yields satisfactory results, it may prove insufficient for certain applications necessitating more comprehensive information about body distribution. Consequently, multi-person 3D full pose estimation emerges as a vital research field with a plethora of applications, including, but not limited to, human-robot interaction. Besides, the torso pose estimator project does not include a reliable and tested method for cross-view correspondence that enables its use in multi-person environments. The multi-person 3D full pose estimator project effectively addresses this matter.

Deep learning serves as an efficacious instrument for learning human physiological priors. However, traditional deep learning solutions call for vast amounts of labelled data, including the model developed in Section 6.2. Addressing this challenge, the work presented in Section 6.3 introduces a deep learning-based method for the 3D multi-pose estimation problem, utilising entirely unannotated data. The approach employs a GNN to discern the views of various individuals in the scene, and an MLP to estimate the comprehensive 3D pose of each person. The only prerequisite for training each network is that every element in the dataset corresponds to a single individual. Both networks use information that can be directly obtained from the RGB images, so the approach only requires conventional RGB cameras.

Experimental results for the skeleton matching model on the CMU Panoptic and ARP Laboratory datasets demonstrate exceptional model performance, as evidenced in section 6.3.2, with near-perfect values across all clustering metrics.

Regarding the 3D pose estimation model's accuracy, in comparison to VoxelPose, the proposed method exhibits marginally lower accuracy values for detected coordinates, with a mean per joint precision error of 26.06mm. However, it is crucial to highlight, as explored in Section 6.3.2, the notable disparity between the detected 2D and the projection of the 3D ground truth in the CMU Panoptic dataset. Employing the projected 3D as a test set, without retraining the network, the mean per joint precision error of the proposed model is 17.84mm (marginally superior to VoxelPose error). This discrepancy renders the evaluation somewhat unfair towards the proposed model, implying that the true precision of the proposed 3D pose estimator is higher. Moreover, the computational complexity of the proposed system is significantly lower than that of VoxelPose, making it an effective solution for real-time applications without the need for annotated data.

The experiments conducted in Section 6.3.2 underscore the merits of training the models with a subset of cameras for inference time. This approach facilitates the system's use in a mobile robot equipped with only two RGB cameras for real-time applications. As demonstrated in these experiments, the system possesses adequate precision for various social robotics applications. In fact, the experiments executed in Section 5.5.4 for human-aware navigation with a robot were conducted employing the 3D HPE developed in this work to detect people's positions in the environment.

Future work will enhance the accuracy of the estimator model by refining the training process through hyperparameter tuning. Additionally, there is a plan to develop models that are trained using data with incorporated intrinsic and extrinsic camera parameters, thereby eliminating the necessity for scenario-specific training. The achievement of this latter goal would make the model robot-agnostic, allowing users to effortlessly mount cameras on the robot, perform calibration, and operate the system without requiring any training.

Chapter 7

Synthesizing Traffic Datasets using Graph Neural Networks

The work presented in this chapter has been adapted from the following publications:

[147] **Rodriguez-Criado, Daniel**, M. Chli, G. Vogiatzis, and L. J. Manso. Synthesizing traffic datasets using graph neural networks. In *International Conference on Intelligent Transportation Systems*. IEEE, 2023.

This chapter transitions from the exploration of indoor sensorized environments in previous chapters to outdoor city spaces, focusing specifically on crossroads monitored by CCTV cameras. These spaces, following the core theme of this thesis, can also be regarded as sensorized environments (specially in smart cities) due to the multitude of sensors present, such as CCTV cameras, air quality sensors, thermometers, and potential actuators like traffic control signals for traffic lights or variable message signs. Building upon techniques developed in preceding chapters, particularly the synergy between Graph Neural Networks (GNNs) and generative models for image generation (as discussed in Chapters 4 and 5), this chapter employs a method where a conditional Generative Adversarial Network (GAN) is conditioned with a GNN. This combined approach aims to generate synthetic, realistic-looking traffic images. These can subsequently be used for training end-to-end deep learning models in traffic applications, including traffic light control and traffic prediction. Another potential application could be generating footage to train individuals for traffic-related roles.

The persistent issue of traffic congestion, especially prevalent in major global cities, is intensified by the escalating number of vehicles, while the navigable urban space remains unchanged. Within this setting, proficient traffic management becomes a critical necessity to reduce travel delays, road accidents, and environmental pollution. Intelligent Transportation Systems (ITS) integrate sensing and communication technologies with automatic control techniques, augmenting the safety and efficiency of transportation infrastructure [4].

Junctions are pivotal points in traffic management as they function as shared physical spaces traversed by numerous vehicles. Effective traffic light control at these intersections can lead to improved traffic flow. Traditional traffic lights, however, are not efficient when they cannot adjust to variable traffic patterns [4]. The progression in machine learning offers a solution through the use of algorithms that deduce optimal policies from raw sensory data. Deep Reinforcement Learning (DRL) allows for the creation of end-to-end models that control traffic light signals directly from CCTV footage. Due to the financial implications and potential hazards associated with collecting real-world traffic data, these models typically depend on simulated data for training, which often leads to difficulties in generalising the acquired knowledge to real-world decision-making. This chapter focuses on bridging the 'sim-real' gap by developing a tool that autonomously generates photorealistic images from 2D traffic simulations (e.g., SUMO [16]) and recorded junction footage.

Garg et al. [53] recently presented a DRL traffic light agent trained on simulated crossroads within a game-like graphical environment, confronting the issue of generalisation through domain randomisation [173]. Their method generates diverse simulated scenarios with varying illumination, perspective, and textures, which enhances model robustness and eases adaptation to real-world conditions. Nonetheless, training on photographic footage eradicates the need for domain randomisation, requiring less training data for the same or superior performance, as the model is trained and evaluated on similar data distribution.

This chapter elucidates the creation of realistic urban images from simulations, with a primary focus on its application in traffic light control. However, the reach of this technology extends far beyond traffic management, encompassing a broad spectrum of exciting

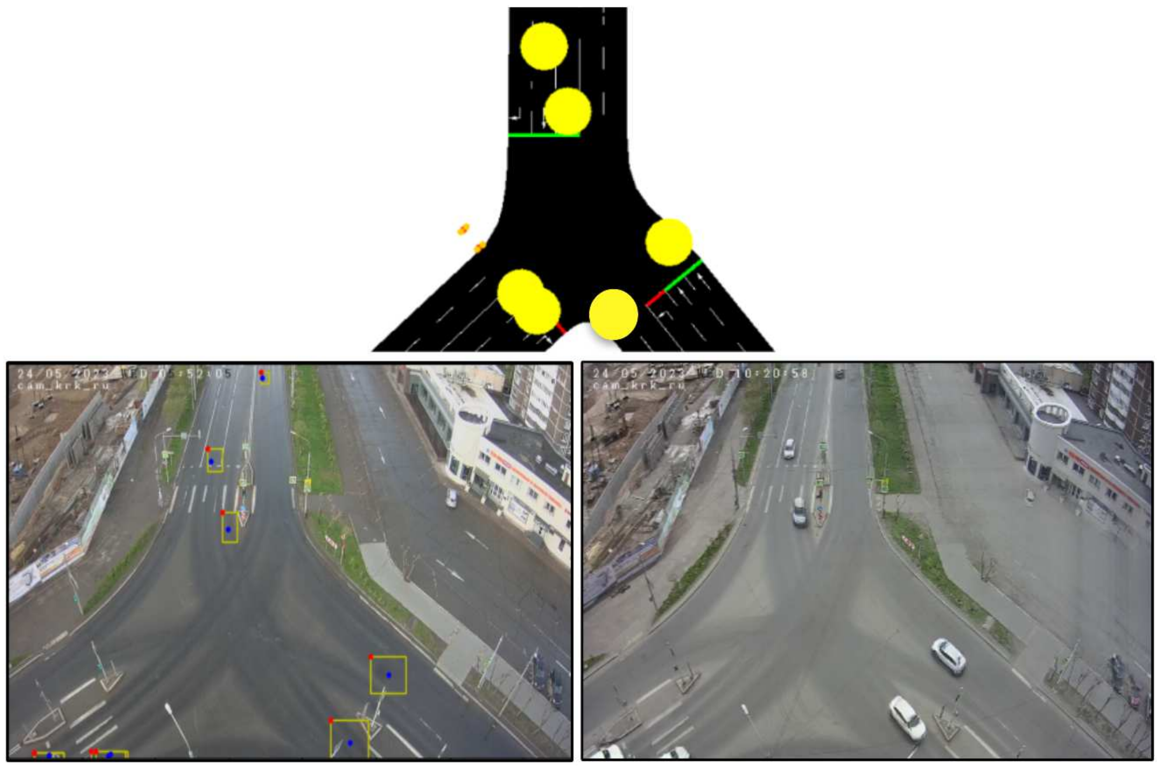


Figure 7.1: The transformation process from SUMO-generated to realistic images. This triptych illustrates the consecutive stages involved in creating a realistic image from the SUMO simulator. The top image provides a bird's-eye view of a junction simulation in SUMO. The bottom-left image presents the corresponding bounding boxes of vehicles in SUMO, adjusted to the viewpoint of the CCTV camera. The image on the bottom right culminates the process by displaying the image generated by our model using the specified bounding boxes.

applications. For instance, Adaimi et al. [2] employ a drone swarm to capture aerial traffic images for training object detection models. By supplementing their dataset with the generation of realistic images, substantial enhancements can be made to model performance and accuracy, as demonstrated in [13]. Such progress offers considerable potential for improving traffic pattern detection and analysis, thereby facilitating more informed decision-making and effective urban planning.

Traffic surveillance also stands to gain from the generation of realistic urban images. Models focusing on vehicle counting or tracking, using camera-based systems [50], can enhance their accuracy and reliability by integrating synthetically generated footage that covers a diverse range of parameters and conditions. Moreover, the creation of synthetically generated footage of traffic scenarios under various parameters and conditions paves the way for immersive and responsive training tools for professionals in traffic control. This provides trainees with a comprehensive understanding of the cause-and-effect relationship between their decisions and the resulting traffic dynamics, far surpassing the limitations of studying historical footage alone.

The transformative impact of this technology lies in its potential to revolutionise training methodologies and decision-making processes across various domains. By generating realistic urban images from simulations, we can uncover novel insights, refine existing models, and enable professionals to devise efficient and effective solutions.

7.1 Background

The ability to synthesize high-quality, realistic images has been a long-standing goal in computer vision. One prevalent application of image generation is data augmentation [9, 126], which is essential in averting overfitting in large-scale deep learning models. Other applications include image completion, style transfer, and resolution enhancement, among others [115, 127, 117].

As explained in Chapter 2, three principal research directions prevail in the literature for generating realistic images: Generative Adversarial Networks (GANs) [60], Variational AutoEncoders (VAEs) [89], and diffusion/denoising models [72]. GANs have gained considerable popularity for image generation due to their capability to produce high-resolution, diverse, and aesthetically pleasing images, as compared to the blurry images often generated by VAEs. While denoising models also achieve high-resolution image generation, they typically demand more computational time compared to GANs and remain relatively nascent due to their recent emergence. For these reasons, a GAN has been selected as the image generation method for this chapter's work.

Certain GAN-based models, specifically conditional GANs (cGANs), can generate novel images conditioned on both training data and a provided condition in the form of an image or label. For example, SPADE [136] can employ segmentation maps as labels to generate images with a realistic appearance. SPADE builds upon the pix2pix model [81], surpassing it by preserving semantic information more effectively in the face of standard normalisation layers (refer to Section 7.2.2 for further details). Nevertheless, these segmentation maps provide conditions only in terms of location and object class. Conversely, text-to-image synthesis models [134, 142] provide more semantic information to the generation model. However, this is at the expense of locality information, and these models are typically larger as they incorporate a natural language processing unit. This chapter proposes a model that can be conditioned with a combination of graphs and segmented images to address these constraints. While segmented images preserve locality information, graphs, when processed by a GNN, can convey more abstract information.

As previously discussed in Chapter 3, a salient advantage of GNNs lies in their capability to process graphs effectively. Graphs are particularly valuable as they intuitively represent the inputs for numerous problems, encapsulating both metric and semantic data along with their intricate relationships. In urban scenarios, these data can include the time of day, weather conditions, vehicle colours, and more. GNNs have also been proven to work well in combination with other models, such as Convolutional Neural Networks, for image generation from graphs. This is exemplified in the generation of cost maps for autonomous robots, as outlined in Chapters 4 and 5.

The method proposed herein generates realistic images of traffic intersections based on an input graph containing positions and colours of various entities, including cars, trucks, buses, and pedestrians, as well as the time of day. The use of GNNs is crucial in the synthesis of urban scenes as they can effectively handle a variable number of entities. Graphs can encode more complex semantic information than segmented images, whilst segmented images can convey the positional information of entities. Once the model is trained, a traffic simulator such as SUMO [16] can generate new scenarios, which the proposed model can readily translate into realistic images.

The **primary contribution** of this chapter is a novel image generation approach that combines a cGAN model (SPADE) with a GNN to generate realistic traffic images from graphs, thereby enabling structured and human-readable conditioning. To my knowledge, this represents the first architecture of its kind. This model can transform simulated traffic crossroad scenarios into realistic images, facilitating the generation of comprehensive datasets with relative ease and minimal cost. These resulting datasets can subsequently serve to train various machine learning algorithms for a multitude of urban traffic applications. Additionally, an application to generate images with vehicles and pedestrians in manually defined positions has been developed to test the model. Details of this tool can be found in Section 7.3.5. More details about this tool can be found in Section 7.3.5. For comprehensive information about the entire project, please follow the URL: `https://vangiel.github.io/projects/traffic.html`.

7.2 Method

The following section provides an overview of the stages incorporated within the methodology. In the initial stage, data collection occurs, leading to the training of the generative model (Section 7.2.2). Detailed information on the model's input components—the graph, the real image, and a segmented map—is given in Section 7.2.1.

Once the training process is concluded, the outputs from the SUMO simulator are transformed into graphs (Section 7.2.3). This allows for realistic images to be generated from these graph inputs, which are used by the image generator model.

The proposed approach represents a significant advancement in the generation of realistic images by effectively leveraging the information encoded in graphs derived from simulations. The rest of this section delves deeper into each stage of the pipeline. The associated code is publicly accessible at `https://github.com/gvogiatzis/trafficgen`.

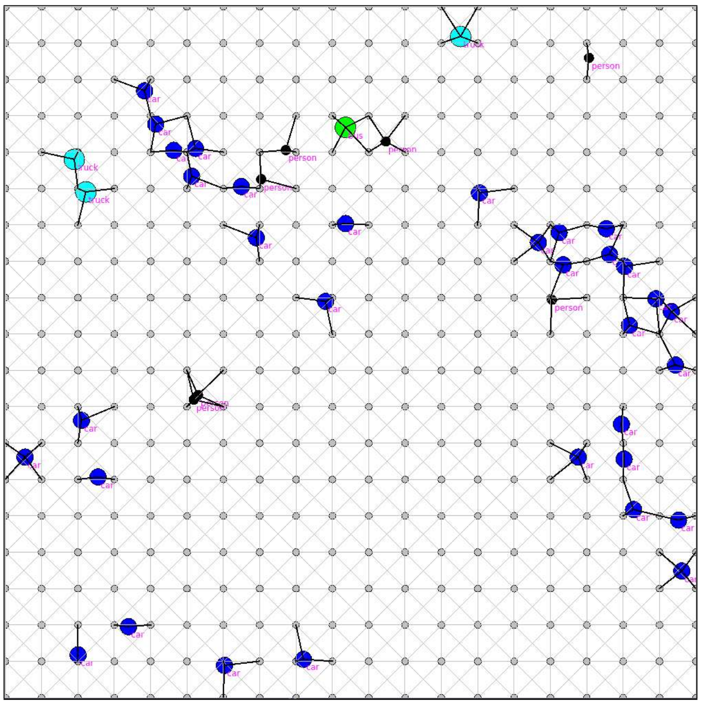
7.2.1 Dataset Creation

The proposed model for image generation takes a three-element tuple as input, consisting of the segmentation map, the graph, and the real image. Fig. 7.2 illustrates a datapoint containing these elements, providing a visual representation of the data used in this study. This section explains the process of extracting the segmentation maps and graphs from real images. The real images utilised for model training in this chapter are sourced from two open-access real-time traffic surveillance cameras for two different traffic junctions. Both are situated in the city of Krasnoyarsk, Russia. The crossroad depicted in Fig. 7.2c will be referred to as **CR1**[1], while the one presented in Fig. 7.1 will be **CR2**[2]. Note that

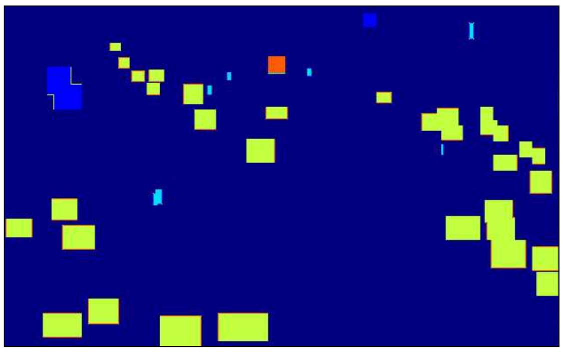
[1]Video streaming URL of the CCTV used: `http://krkvideo14.orionnet.online/cam1560/embed.html?autoplay=true`

[2]Video streaming URL of the CCTV used: `http://krkvideo5.orionnet.online/cam1487/embed.html?autoplay=true`

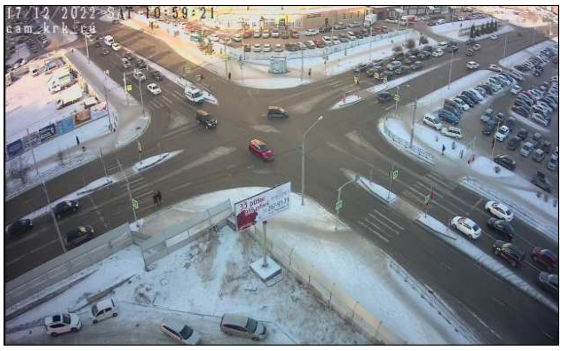
generating images for a different junction requires the collection of new data, specific to that particular setting, followed by retraining the model to accommodate the new input. In total, 10,952 images were gathered from **CR1** videos and 11,173 images from **CR2** videos, in a period of 24h each.



(a) Illustration of an input graph. Nodes in light grey represent grid nodes, while those in dark blue signify cars. Green nodes correspond to buses, light blue nodes denote trucks, and small black nodes represent pedestrians. The grid in this image has a resolution of 20 × 20 nodes.



(b) Labelled segmented image. This illustration depicts the positions and sizes of the classes identified by *YOLOv7* in the real image, distinguished by varying colours.



(c) Real image. This image was captured from a CCTV camera positioned at a crossroad in Krasnoyarsk, Russia.

Figure 7.2: Illustration of a dataset data point from CR1, comprising three elements. This figure incorporates the three constituent elements of the dataset: the input graph, segmented image and real image.

Segmentation Map Generation

A segmentation map in this context is a type of image that represents different segments or regions of the original image, with each segment corresponding to a specific class present in the image. Each pixel in a segmentation map is assigned a label that identifies the category of its corresponding pixel in the original image. In a visual representation of a segmentation map, each unique label is represented by a unique colour, making it easy to visually distinguish between different segments or regions of the image.

Generating the segmentation map is a straightforward process. An object detection model, namely *YOLOv7* [185], is applied to the real image, yielding bounding boxes for several classes: cars, buses, trucks, and pedestrians. It should be noted that any other object detection model could serve the same purpose. Importantly, the number of classes can be effortlessly extended within the range of classes detectable by *YOLOv7*, if required. However, it's important to acknowledge that the detection model is not flawless; it might yield false positives and false negatives, which could impact the final generated image. The potential for improved detection accuracy in future work might significantly enhance the overall results.

Utilising the normalised coordinates and dimensions of the bounding boxes for each detected object, a segmented map can be created in which each pixel value corresponds to an integer representing the class using a one-hot encoding. If the detection of two objects overlaps, the class from the latest detection is allocated to the pixels within the intersection—a design choice that may be modified in future work. As the example model generates five distinct classes (including the image background), the pixel values are in the range $[0, 4]$. Fig. 7.2b shows an example of a segmented image drawing the classes with different colours.

Graph Generation

The creation of graphs is a critical and intricate step that offers extensive customisation flexibility to the designer, as seen in previous chapters. Due to the wide range of potential graph representations for this application, numerous variants have been explored. For brevity, this section will only discuss the two design strategies yielding the most favourable results, which only differ in how objects' colours are represented. The first design, referred to as the *clustering-colours* graph, produces the best outcomes in terms of the quality of the generated images. On the other hand, the second design, named *discrete-colours* graph, while resulting in slightly inferior performance, enables the user to condition the vehicle colours using a discrete colour palette. This will be examined in greater detail in Section 7.3. Both graphs are identical, except for a minor variation in the nodes' features, as explained subsequently.

Node feature vector h (dimension 31 or 19)			
Boxes	**Classes**	**Time**	**Colour encoding**
x y w h	bus truck car person grid	sin cos	clusters-colours (dimension =20)/ dircrete-colours (dimension=8)

Table 7.1: Representation of the feature vector structure for each node in the graph.

The creation of the topology of the two aforementioned graph types is the same. First, a graph is created in which each node represents an entity detected by the object detector, *YOLOv7*. A lattice of nodes is generated (in the same fashion seen in Chapters 4 and 5), with each node representing a spatial position within the image. This grid is crucial for conditioning SPADE, as explained in Section 7.2.2. In the final step, both graphs are merged by connecting the closest entity-representing nodes to the nearest nodes in the grid within a specified radius, using the image coordinates. Both the grid density and the connectivity radius are adjustable hyperparameters. Various values were assessed to achieve the optimal balance between accuracy and efficiency. Fig. 7.2a provides an example of the final graph's topology, employing a grid resolution of 20×20 and a connectivity radius of 1 grid hop, assuring 8-connectivity.

As mentioned, the vector of characteristics of the nodes differs for the two types of graphs for the colour representation. The feature vector is classified into four sections, as delineated in Table 7.1. The first three sections, which are shared across both graph types, encompass: the coordinates and dimensions of the object bounding box; a one-hot encoding vector of length 5 indicating the node class (bus, truck, car, person or gird); and the encoding of the time of day using the sine and cosine.

The final section of the vector, which we will refer to as "visual features", diverges between the two graph types:

- For the *clustering-colours* graph: This includes a 20-element vector indicative of the object's primary colours. This vector comprises clusters of the top 5 most predominant colours within the bounding box, encoded in RGB. Along with the three RGB numbers, each cluster includes an additional number with the counts normalized using a *softmax* function. This configuration results in a total feature vector length of 31.

- For the *discrete-colours* graph: Here we employ a one-hot encoding of length 8, representing the detected vehicle's colour. The colour palette includes black, white, red, lime, blue, yellow, magenta, and grey. Colour detection consists in averaging the top 3 most predominant colours in the bounding box and calculating the Euclidean distance to each palette colour based on their RGB coordinates. The colour with the shortest distance is chosen. Given that averaging shifts the colour closer to grey, the mean value must exceed a certain distance threshold for the specific case of selecting the grey colour. With this option, the total length of the feature vector is 19.

By integrating these features into the graph nodes, we enrich the SPADE generator with information about the entities' colours and the time of day. The *clustering-colour* graphs provide more detailed colour information leading to better results, while the *discrete-colours* graph enables a straightforward indication of the entity colour during inference time. The benefits of employing the *discrete-colours* graphs become evident when utilising the demonstration tool presented in Section 7.3.5.

7.2.2 Model

This section details the combination of architectures for generating the images. The final model consists of a modification of SPADE architecture including a GNN to condition the generator with a graph. The graph can provide richer information to the generation model allowing the creation of more complex conditions which are reflected in the final image. First, the vanilla structure of SPADE is shown. The next subsection explains the architecture embedded in SPADE that takes graphs as inputs. Finally, we provide a global version of the final model pipeline for generating images from graphs.

GAN Model, SPADE

To generate images from semantic masks, SPADE layers transform segmentation masks into feature maps γ and β by first projecting the mask onto an embedding space that we will call condition volume $\boldsymbol{\omega}$. Then, this volume is fed through two convolutional layers to get the feature maps. The generated parameters γ and β, which are tensors with spatial dimensions, are multiplied and added to the normalised activation from the previous layer h, element-wise. Thus, the activation features $h_{n,c,h,w}$ are normalised and transformed as follows:

$$h'_{n,c,h,w} = \gamma_{c,h,w}(\boldsymbol{\omega})\frac{h_{n,c,h,w} - \mu_c}{\sigma_c} + \beta_{c,h,w}(\boldsymbol{\omega}), \tag{7.1}$$

where the indexes (n, c, h, w) refer to the batch size, the number of channels, the height and the input width respectively. The parameters μ_c and σ_c denote the channel-wise mean and standard deviation of the input feature map h.

The generator incorporates multiple ResNet blocks [68] with upsampling layers. The semantic map is downsampled to align with the resolution required for learning the modularization parameters γ and β, as each residual block operates at a distinct scale. In contrast, the discriminator does not employ SPADE and follows the pix2pixHD [187] discriminator approach, based on PatchGAN [81], which inputs the concatenated segmentation map and the input image.

The main modification introduced in this chapter for the generator consists of generating the condition volume $\boldsymbol{\omega}$ using a combination of a GNN and transpose convolutional layers as explained in Section 7.2.2. Now, instead of downsampling the semantic map, it is $\boldsymbol{\omega}$ that is rescaled to match the needed resolution in each layer. With regards to the discriminator, the change implies the additional concatenation in the channels' dimension of information coming from the GNN to make the GAN symmetric (see Section 7.2.2).

Condition Model

As already mentioned, the input data (*i.e.*, vehicles, pedestrians and other contextual information such as the time) are combined with a 2-dimensional lattice to form an input graph

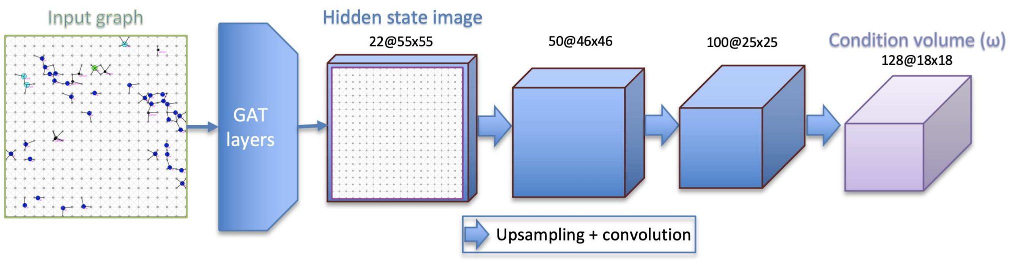


Figure 7.3: This illustration shows the process of generating the condition volume $\boldsymbol{\omega}$, which is used to condition the image generation of SPADE. The graph is processed by 3 GAT layers, and then the lattice of nodes is filtered from the output graph forming a hidden state image. This image is subsequently fed into an array of upsampling and convolutional layers to fabricate the final condition volume. Each volume depicted in the image has its dimensions written at the top, where the first number denotes the channels, and the succeeding pair specifies the height and width.

(Fig. 7.2a). As with most GNN layers, GAT layers output graphs with the same structure as their input graph but different node embeddings as seen in Chapter 3. The input graph is processed by 3 GAT layers, producing a graph with the same structure but adequate to condition SPADE after being filtered and processed. This is done by training the GAT layers to embed the entities' information into the lattice sub-graph so that the lattice can be converted into an image-like format creating a latent image. This is done by creating a pixel for each of the nodes in the lattice and using the features of the nodes as the channels of the image. Finally, this latent image is the input of 4 transpose convolutional layers that generate the desired condition volume $\boldsymbol{\omega}$. This pipeline is depicted in Fig. 7.3.

Final Model

The final model adds the condition module to the SPADE blocks of the generator and also includes some modifications to the discriminator of SPADE to make the network symmetric. Fig. 7.4 is a diagram of the final model pipeline with the generator and the discriminator of SPADE. The data flow starts with the segmentation mask as input of the generator, then the condition module uses the graphs to yield the volumes that are also fed to the generator. The generator produces a fake image that is concatenated in the channel dimension with the corresponding mask and the latent image of the condition module. Finally, the real image is concatenated with the mask and the latent image in the channel dimension, and then the real and fake images are combined in the height dimension. This generated fake-and-real volume is the final input of the discriminator. Note that one fake image is generated every two real images fed to the discriminator as done in the original version of SPADE.

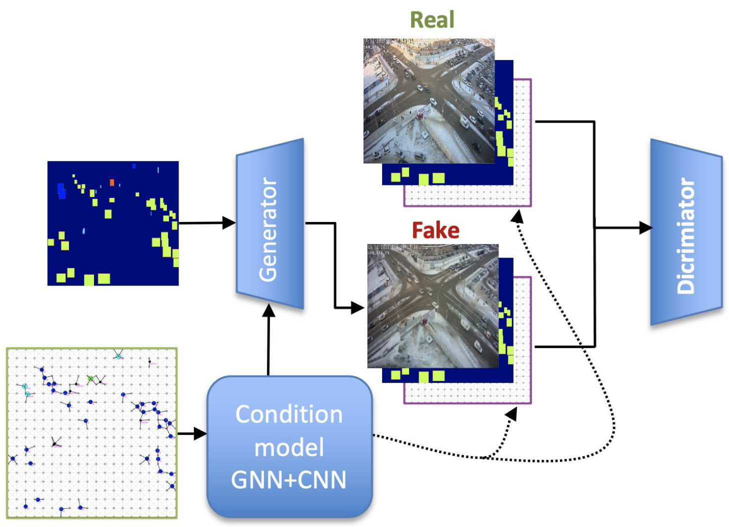


Figure 7.4: Complete model pipeline. The schematic depicted herein illustrates the integration of the condition model within the SPADE architecture. The condition model accepts the graph as input, producing the condition volume utilised by the generator. Moreover, the hidden state image stemming from the condition model is concatenated in the channel dimension to the discriminator's input.

7.2.3 From SUMO to Graphs

One of the main aims of this work is to drive image synthesis from traffic simulations. One can then apply various actions to the traffic system (e.g. traffic control decisions) and observe their effect through visual footage generated in response. This not only bridges the gap between trial and error but also fosters the training of decision-making entities (including AI and humans) using footage generated in reaction to the decisions made. An instance of this loop closure, using SUMO [16] as the driving simulator, is demonstrated herein.

The first step entails constructing a topologically accurate representation of the traffic junction network within SUMO. This can be readily accomplished using scripts capable of importing geometry and network structure from OpenStreetMap (OSM) data. The second step necessitates the establishment of a correlation between 2D points on the SUMO simulation and points on the real-world junction. Assuming geometric accuracy of the OSM data, this correlation would simply be a 2D homography with 8 degrees of freedom. However, significant discrepancies are often encountered between the simulation geometry and the real-world junction.

In this chapter, the approach adopted involves defining individual traffic lanes in the junction as cubic splines. This can be efficiently completed within a few minutes by clicking through points in the junction images, a method that has proven effective in practice. Fig. 7.5 depicts the lane designer GUI with the SUMO lane selector (left) and the cubic spline lane editor (right). Within each lane spline, corresponding waypoints between SUMO and the real world can be defined (e.g., the point where vehicles halt for the red light and other clearly demarcated landmarks). The outcome of this process is a reliable mapping between points on each SUMO lane and the corresponding points on the real junction.

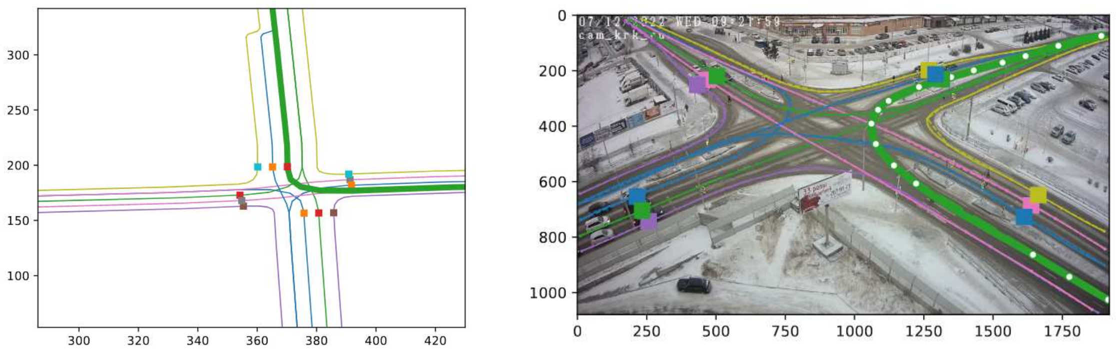


Figure 7.5: This illustration exhibits the interface of the lane designer application, juxtaposing the SUMO lane selector (left) with the cubic spline lane editor (right). It facilitates the definition of corresponding waypoints between the SUMO environment and real-world images for each lane spline.

The final stage of this process involves determining vehicle bounding boxes for each location within the junction. These bounding boxes subsequently aid the creation of graphs (as elaborated in Section 7.2.1) which serve as inputs for our image generation model. To procure these bounding boxes, a spatial bounding box distribution is derived utilising histograms. Fig. 7.1 illustrates the progression from a SUMO frame (left) to a set of bounding boxes defined on the actual junction image (middle), culminating in a synthesised CCTV frame (right) featuring the road background and correctly positioned vehicles.

7.3 Experimentation and Results

This section presents an in-depth examination of the experiments conducted with the generation model, along with the resultant findings. The final subsection encompasses a description of the demonstration tool developed for manual testing of the model.

7.3.1 Implementation Details

All experiments were carried out utilizing an NVIDIA RTX A6000 GPU that has a memory capacity of 48GB. All the models were trained with image resolutions set at 640x640 pixels, whereas YOLOv7 was used to detect bounding boxes from images with a resolution of 1280x1280 pixels. A batch size of 12 was employed for the training phase, and this was increased to 24 during testing.

7.3.2 Dataset and Metrics

Three different models were trained with the **CR1** dataset: the standard SPADE version, and the combination of the GNN and SPADE for the two types of graphs. These models were trained using an identical training set consisting of $10,322$ images, graphs, and segmentation maps. Each model was evaluated using three distinct metrics on a test set comprising 630 data points, following the same metric system utilised by SPADE [136]. To quantify segmentation accuracy, we used mean Intersection-over-Union (mIoU) and pixel accuracy (Accu.). The Frèchet Inception Distance (FID) [71], on the other hand, was utilised to evaluate the discrepancy between the distributions of synthetic and real images.

Models	**FID**	**mIoU**			**Accu.**		
		Cars	People	Trucks	Cars	People	Trucks
SPADE	176.32835	**0.63349355**	**0.3243129**	0.099982	**0.70595584**	**0.59278189**	0.13323193
cluster-colours	**149.88987**	0.53274998	0.21035392	**0.27845053**	0.69294361	0.24580686	**0.34741428**
discrete-colours	154.56592	0.50205496	0.16974847	0.01920121	0.66411157	0.20146478	0.02427287

Table 7.2: Results for the three different models evaluated in the proposed metrics.

The mIoU and pixel accuracy metrics for each class were calculated, excluding the 'background' due to its disproportionate size relative to other classes. The 'buses' category was also omitted due to a lack of sufficient images within the training dataset to facilitate a decent generation of these vehicles. The performance against these metrics is summarised in Table 7.2.

The analysis of the results reveals that the proposed models demonstrate a significant enhancement in the FID score compared to SPADE, with the *cluster-colours* model producing superior outcomes for this metric. Although the mIoU and pixel accuracy of the proposed models are generally slightly lower than SPADE results, they remain competitive, with the *cluster-colours* model producing superior results for trucks. Bear in mind that the

FID is the most critical measure for this study, as it evaluates the realism of the generated images, whereas the other two metrics pertain more directly to image segmentation models. Nevertheless, the mIoU and pixel accuracy were included as these metrics were employed in the original SPADE paper.

It is also worth highlighting that the additional computational burden of the *cluster-colours* and *discrete-colours* models over the basic SPADE model is marginal, adding only 10.42% and 10.28% more parameters, respectively. Consequently, the computational speed and memory usage remain comparable. The training for the proposed models and SPADE for the specified batch size and number of images took approximately 3.5 days. The generation of images for the test set takes approximately 2 minutes for all the models excluding the time needed to load the data.

7.3.3 Visual Results

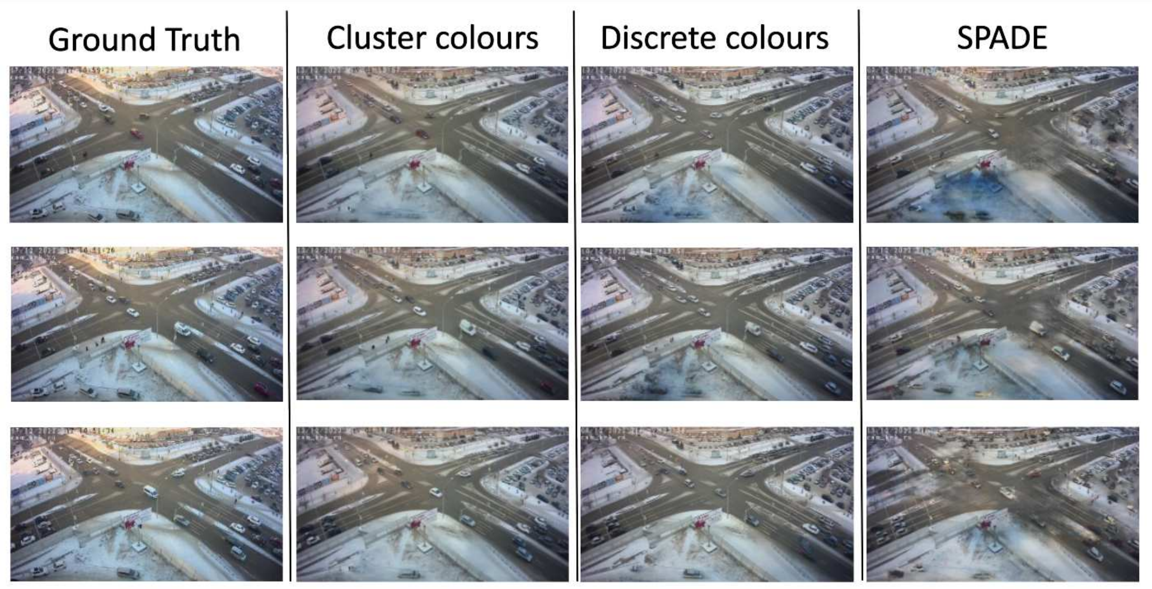


Figure 7.6: This illustration presents the results for three distinctive frames from the test dataset, each in separate rows. The leftmost column constitutes the ground truth images, succeeded by images generated by the *cluster-colours*, *discrete-colours*, and SPADE models respectively. As observable, the *cluster-colours* model is the most proficient at preserving vehicle colours, whereas the SPADE-generated images exhibit the poorest quality.

In order to interpret the above metrics more tangibly, several frames generated by each model are presented in Fig. 7.6. Each row in this figure represents a different frame from the dataset, where the first image corresponds to the ground truth derived from the test set, and the subsequent images are the outputs generated by the *cluster-colours*, *discrete-colours*, and *vanilla* SPADE models, in sequence.

A close inspection of the results reveals that the *cluster-colours* model is proficient at effectively reconstructing the colours of the majority of vehicles present in the original images. However, the *discrete-colours* model tends to struggle in generating vibrant colours, although it performs adequately with white, black, and grey vehicles. This issue could potentially be attributed to the method employed to discretise the colour palette, a point that could be optimised in future work. In contrast, images synthesised by the vanilla SPADE model display vehicles with arbitrary colours, and they often introduce artefacts into the majority of frames, rendering them less visually appealing and realistic.

7.3.4 Time Conditioning

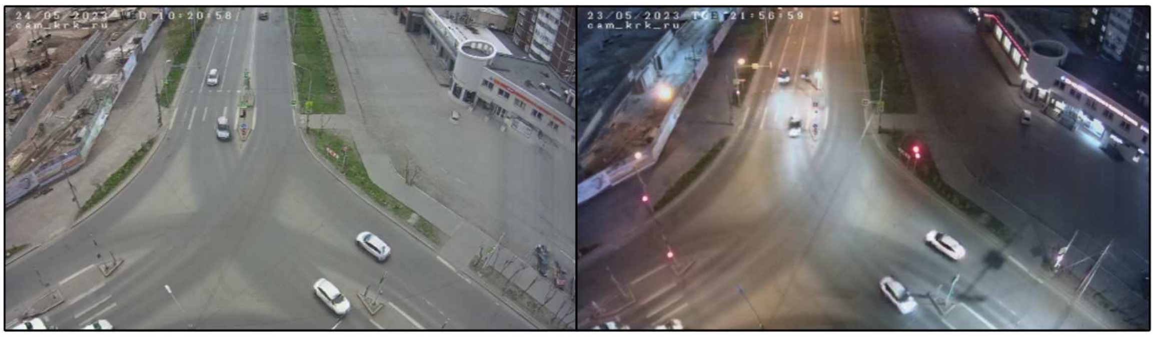


Figure 7.7: Two examples of images generated at different times of the day. These images were generated using the demo tool in Section 7.3.5. It is readily apparent that the daytime-generated image (left) possesses more illumination, whereas the nocturnal counterpart (right) exhibits a darker ambience, authentically simulating the respective time frames.

As previously discussed, the proposed model in this study is also capable of generating images conditioned at different times of the day. Fig. 7.7 demonstrates this capability, featuring two images generated by the *discrete-colours* model corresponding to daytime (Fig. 7.7 left) and nighttime conditions (Fig. 7.7 right).

7.3.5 Interactive Tool

To evaluate the image generation model under various conditions, a tool with a visual interface was designed, as illustrated in Fig. 7.8. The graphical user interface (GUI) displays two distinct frames. On the left frame, the user can draw various bounding boxes, each denoting the placement of entities to be generated.

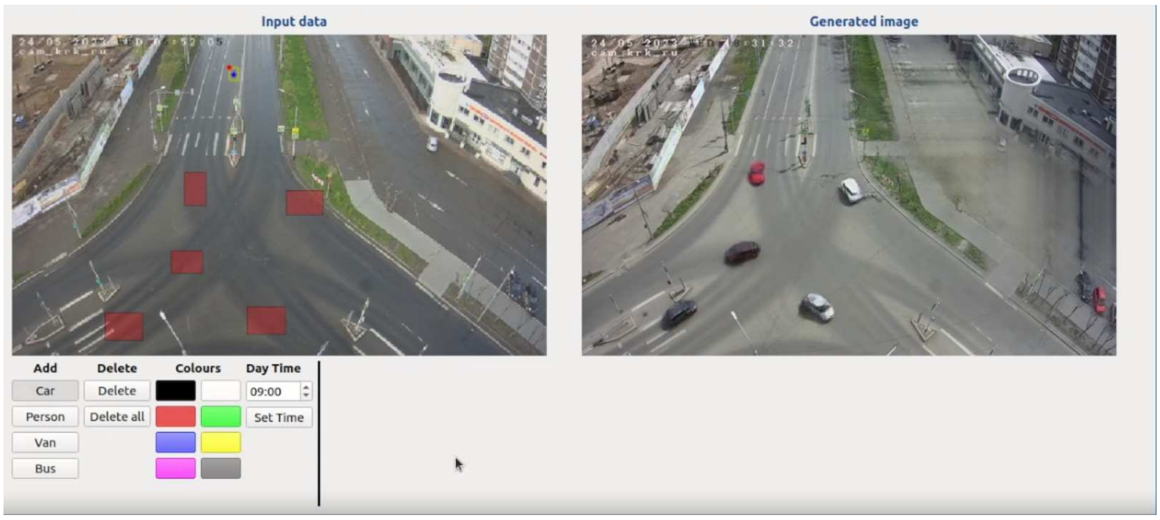


Figure 7.8: Graphical user interface of the interactive tool. The application's interface is divided into two principal frames: the left frame facilitates the entry of user input data, while the right frame displays the generated image. The toolbox for choosing vehicle types, colours, and time of day – parameters to condition the image generation – is located in the lower-left corner. Users can sketch an arbitrary number of bounding boxes within the left frame, indicating the desired positions of entities to be generated.

The entity type, its colour, and the time of day can be specified using the tools button, situated at the lower-left corner of the GUI. The right frame exhibits the image produced by the model, reflecting the entities and conditions specified within the left frame. A new image is generated every time a modification is done in the left frame.

7.4 Conclusions

The synthesis of realistic images from simulated ones is highly beneficial and presents a myriad of applications, where data augmentation stands out. The present chapter has presented the development of a tool that is capable of generating traffic images with a realistic appearance from a simulator by merging a GAN-based model, SPADE, with a GNN for conditioning.

This tool enables the production of large datasets with relatively less effort, vital for training computationally demanding deep learning models that comprise numerous parameters (among other applications), thereby preventing the overfitting of the training data. This results in enhanced generalisation to new scenarios. Section 7.3 illustrates the effectiveness of the proposed model in comparison to the unmodified version of SPADE, with only a minor increase in computational complexity.

The presented model is not only capable of generating realistic images but also conditioning features of the generated images using semantic information, namely the colours of the vehicles in the image and the time of the day. Section 7.3.5 shows a tool with a GUI that enables the user to produce images by manually setting the stated conditions along with the positions of the entities to be created.

Looking towards future research, the plan is to train the model with additional data and utilise more than one consecutive frame as input for the model to ensure stability in videos. These improvements are likely to enhance the quality of the generated images. Another goal is to expand the scope of experimentation by introducing more classes for the model to generate and integrating additional conditions such as weather. This expansion would necessitate improvements in the detector model. An intriguing prospect for subsequent research involves substituting the current cGAN with a diffusion model. Given the noteworthy results achieved by these models in recent years, this approach could yield valuable insights.

Chapter 8

Conclusion

This thesis presents several efficacious applications of deep learning within graph domains, specifically in the context of indoor and outdoor sensorised environments. As our society witnesses a proliferation of 'smart' cities, homes, and public spaces, these sensor-rich environments are becoming more prevalent. An emerging paradigm within the sphere of deep learning in graph domains, Graph Neural Networks (GNNs), has proven to be an exceptionally potent tool within this field, as substantiated throughout this work. The models and tools developed herein offer pragmatic solutions to real-world challenges posed by our progressively sensorised world.

The contributions within this thesis can be categorised into three primary domains: cost maps and discomfort scores for human-aware navigation with robots, human pose estimation in indoor sensorised environments, and image generation for traffic applications. The first two categories can significantly enhance the functionality of assistive robots, whether in the homes of elderly or disabled people, in public settings such as events and restaurants, or even for delivery robots. Conversely, the last application provides a valuable tool for generating realistic traffic images, supporting a variety of deep learning applications in traffic management. This thesis not only underscores the impact of the developed projects but also propels new research avenues within the paradigm of GNNs and sensorised environments.

The principal questions this thesis aims to address are stated in Chapter 1. We have underscored numerous advantages of GNNs when employed in sensorised environments, outperforming conventional deep learning models as outlined in Chapter 3 and empirically validated in the contributions of this thesis. The experiments conducted in the present work illustrate the unique capabilities of GNNs, exploiting their inductive bias for graph-structured data and relational information to yield superior performance than state-of-the-art deep learning techniques within this niche field.

The thesis posits that, when applied efficiently, GNNs can surpass classical methodologies for specific applications within sensorised environments. The efficacy of these networks is manifest in tasks such as data fusion from multiple RGB cameras, effective utilisation of data relationships, and the efficient extraction of semantic data from graphs. This is exemplified in the generation of cost maps that account for interactions between people, objects, and robots (Chapters 4 and 5), where traditional deep learning models grapple with relationship identification. Additionally, the swift estimation of 3D human poses through the aggregation power of GNNs, particularly when fusing data from multiple camera sensors, is demonstrated in Chapter 6. The incorporation of semantic data into graph features also presents a distinct advantage, as proven by its utilisation in conditioning the generation of images (Chapter 7). The inherent flexibility of GNNs allows for enhancements in specific cases. For instance, altering input graphs or integrating them with other artificial neural network architectures may lead to improved outcomes.

Despite GNNs' more universal applicability compared to MLPs and CNNs, they often suffer from slower implementation owing to their more complex calculations. The contributions of this thesis have exploited the strengths of both worlds by creating hybrid architectures to address unique challenges. Collectively, the experiments and methodologies developed provide a comprehensive overview of the efficient application of GNNs in sensorised environments.

Over the course of these almost four years, a continuous learning journey has unfolded, each stage of which is mirrored in the chapters of this thesis. I commenced my doctoral endeavour by joining ongoing projects concerning the estimation of disruption scores, an experience that was not only enlightening but also allowed me to make significant contributions. Following this, I collaborated with fellow researchers in the specialised field of human pose estimation. Ultimately, the accumulation of knowledge and skills enabled me to conduct independent research in the final phase of this thesis, focusing on the generation of realistic traffic images.

8.1 Summary of Contributions

In this thesis, we have proposed a multitude of novel contributions, which answer the research questions stipulated in Chapter 1, Section 1.1. The key contributions have been categorised into three key areas:

Contributions to Human-Aware Navigation with Robots

- **SNGNN-2D** is introduced in Chapter 4. This is a hybrid architecture that blends GNNs and CNNs to create two-dimensional cost maps for planning human-aware navigation trajectories. The robustness and performance of this model are confirmed by the experiments detailed in section 4.4.

- A novel efficient approach enabling the combination of GNNs and CNNs, is proposed. This approach consists of an additional grid of nodes introduced into the GNN graphs capturing the 2D information of the scene. Subsequently, this grid of nodes is converted into an image that can be processed by a CNN to generate the final output.

- A novel dataset, The **SocNav2 dataset** is unveiled in Chapter 5, offering short video sequences of a 3D environment that encapsulates the dynamics of a room with a mobile robot (refer to Section 5.2). In addition to including the velocities of the entities involved, this dataset also includes types of interactions between humans and human-objects. The SocNav2 dataset is a valuable asset for training machine learning models in the realm of human-aware navigation and social robotics. Furthermore, it serves as an effective benchmark for assessing the performance of such models. This dataset contributes towards fostering further research and development within these fields.

- In Chapter 5, Section 5.4, **SNGNNv2** is introduced, an improved model capable of generating discomfort scores from dynamic scenarios, addressing the primary limitation of *SNGNN* [118]. SNGNNv2 adopts a similar approach to its predecessor but with notable enhancements. It considers two distinct scores to assess different facets of social navigation, and the model is trained utilising dynamic scenes in which humans and the robot are in motion, addressing the primary limitation of *SNGNN*. A novel method for creating graphs that consider the dynamics of the environments is proposed to work with this new model (Section 5.3 and Section 5.5.2).

- **SNGNN-2Dv2** is presented, utilising *SNGNNv2* to generate a new dataset of images, allowing the creation of disruption maps from dynamic data in Section 5.5.

Contributions to Perception in Indoor Sensorised Environments

- In Chapter 6, a model that estimates the 3D pose and orientation of the human torso using GNN is detailed. This represents a pioneering approach to predicting pedestrian orientation.
- A GNN-based solution is described for matching various 2D poses from multiple cameras in scenarios with a variable number of people. This solution overcomes the limitations identified in the previous contribution.
- A model capable of inferring the 3D keypoints of detected humans is introduced. It uses self-supervised learning to minimise the difference between the detected 2D keypoints and the estimated pose re-projections. Furthermore, the model demonstrates adaptability to mobile robots across different scenarios without the need for retraining, provided only fixed onboard cameras are used (Section 6.3.2).

Contributions to Image Generation for Traffic Applications in Outdoor Sensorised Environments

- In Chapter 7, a novel image generation method combining a cGAN model (SPADE) with a GNN is presented. This method is capable of generating realistic traffic images from graphs, offering structured and human-readable conditioning of the generated images. The model can be used for transforming simulated traffic scenarios into realistic images, thereby facilitating the creation of comprehensive datasets for various urban traffic applications such as traffic light controls or traffic forecasting.

- An application for generating images with vehicles and pedestrians in user-defined positions is discussed. This tool serves to test the model, with further details available in Section 7.3.5.

8.2 Limitations, Directions and Future Work

Despite the substantial contributions detailed in the preceding sections, further research opportunities persist within each of the three categories outlined.

In relation to the work on human-aware navigation for robots developed in Chapters 4 and 5, the prospective scope for investigation involves user profiling and personalisation of the most recent models, namely *SNGNNv2* and *SNGNN-2Dv2*. This could encompass consideration of human activities and gaze patterns, as exemplified by studies such as [35]. Furthermore, an emergent line of research explores strategies for correlating the output of *SNGNNv2* (pertaining to questions Q1 and Q2) with the robotic driving control mechanism. Although an end-to-end solution is conceivable, it presents challenges in obtaining labelled examples and modulating the final control action. A compelling alternative may lie in using the GNN output as an additional constraint, which a Model Predictive Controller can meet [128]. To improve the performance of the models, future work will also focus on training the models to estimate interactions based on previous human behaviour and movements instead of relying on third-party models for detecting interactions. A particularly intriguing endeavour would be to integrate *SNGNNv2* (or a similar model) within a reinforcement learning framework to produce task-specific rewards.

Focusing on the pose estimation with RGB cameras presented in Chapter 6, future endeavours aim to optimise the estimator models' precision through rigorous hyperparameter tuning. Employing a robust third-party 2D detector such as MoveNet [12]—capable of discerning full skeletons—would likely bolster the overall pipeline accuracy. Additionally, an initiative is in place to devise models that incorporate intrinsic and extrinsic camera parameters within their training data, thereby obviating the need for scenario-specific training. Realising this ambition would render the model robot-agnostic, thereby facilitating camera installation on the robot, simplifying calibration, and eliminating the need for supplementary training. Another significant enhancement for the 3D full pose estimator would involve tracking the 2D poses in the cameras and querying the matching network only when the tracker is lost. This strategy could significantly reduce the number of queries to the model, thus enhancing its operational speed.

Lastly, the realistic traffic images generation project presented in Chapter 7 arguably harbours the most promising future prospects, given its extensive potential. Future directions include augmenting the model's training data and employing multiple sequential frames as model input—enhancements expected to improve the stability of generated images. A further ambition involves encompassing additional generative classes and integrating varying conditions such as weather scenarios. An intriguing avenue for ensuing research could entail replacing the existing cGAN with a diffusion model. Given the impressive outcomes achieved by such models in recent years, this strategy might uncover valuable insights.

Appendices

Appendix A

ARP Laboratory Experimental Setup

This appendix delineates the specifications of the experimental setup implemented within the Autonomous Robotics and Perception (ARP) Laboratory at Aston University, as used in the experimentation of Chapters 5 and 6. The setup encompasses the robot, an array of wall-mounted cameras, and the communication system interlinking these components.

In addition, the appendix provides specifications for the machine learning server at Aston, which has been primarily employed for training the majority of models designed throughout this thesis.

A.1 Robot

The robot utilised for the experiments is the differential RB-1 base from Robotnik, the specifications for which can be located in Section A.1.1. Upon this base, we have engineered a body that incorporates an additional assortment of sensors and actuators, alongside a tactile screen to facilitate user interaction. These components are detailed in Section A.1.2.

A.1.1 Robot RB-1 Base

The RB-1 base mobile robot, as described by Robotnik, is designed specifically for the development of indoor logistics applications. This platform has the capacity to transport varying payloads and materials or to integrate alternative systems such as a robotic arm or torso. For our purposes, we have engineered a torso atop the robot as detailed in Section A.1.2.

The version of the RB-1 base acquired for the ARP laboratory is equipped with a localisation and navigation laser, operational over distances ranging from 5 to 30 metres, as well as an RGBD sensor for obstacle detection - namely, the UST-20LX sensor and the Orbbec Astra sensor, respectively. These features enable the robot to halt, circumvent obstacles, or seek an alternate path as required to reach the succeeding waypoint.

The robot's software comprises a control system, a laser-based localisation system, a navigation system, and a rudimentary HMI user interface. All Robotnik robots are fully customisable and built on ROS. Further information about this base and its sensors can be obtained from the manufacturer's webpage: `https://robotnik.eu/products/mobile-robots/rb-1-base-en/`

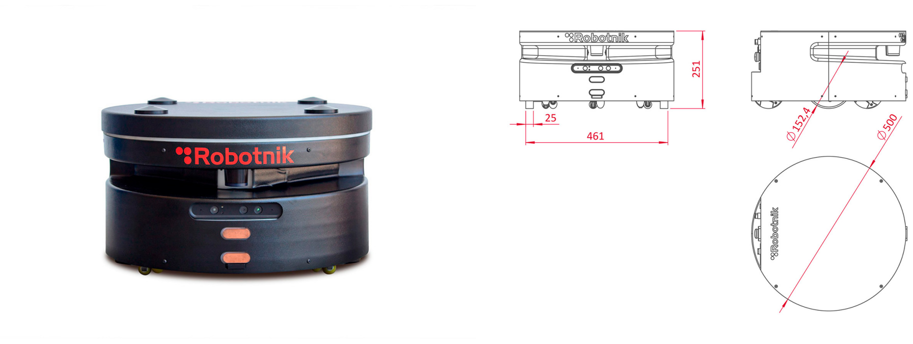


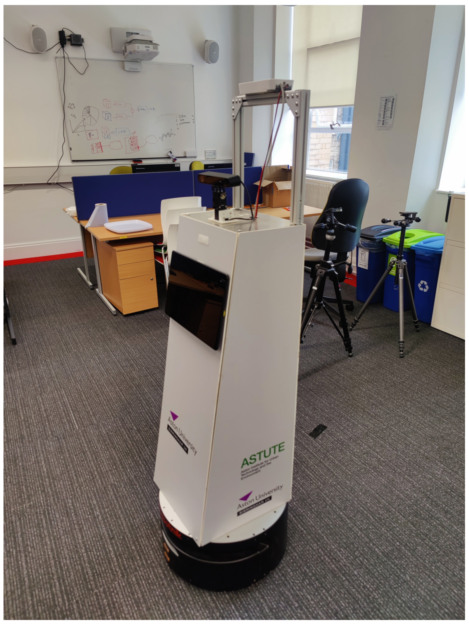
Figure A.1: Dimensions of the RB-1 Base from Robotnik.

Figure A.1 provides an image of the RB-1 base along with its dimensions. Additionally, the robot is supplied with a PS4 controller, facilitating manual operation when necessary.

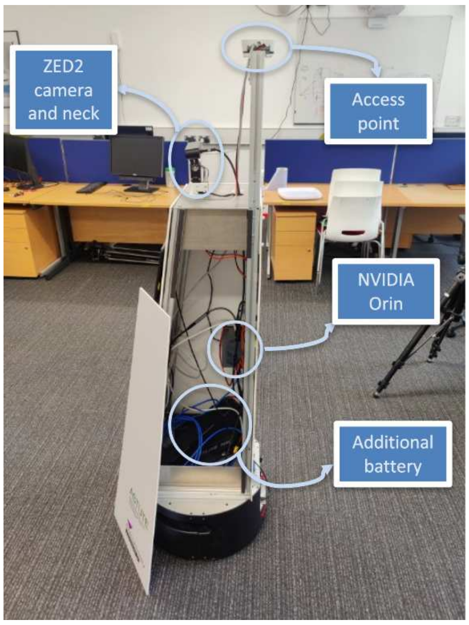
A.1.2 Robot's Body

A customised structure was engineered atop the RB-1 base to engender a more user-friendly aesthetic, given the robot's intended use in human-aware navigation experiments. This structure also houses several sensors, actuators, and an additional computer - an NVIDIA Jetson Orin.

Figure A.2a illustrates the final structure affixed to the RB-1 base, comprising a metallic frame cloaked in FoamX sheets. These sheets are magnetically attached to the structure, facilitating easy removal. A tactile screen, connected to the Jetson Orin, is integrated into the front of the robot to enhance interactivity. The body of the robot incorporates various components, which are more distinctly visible in Figure A.2b where one side of the structure is detached.



(a) Image of the structure with the front screen.



(b) Detailed image of the robot's body components.

Figure A.2: Images of the robot's body constructed on the RB-1 Base.

The components include an access point located on top of the structure for communication with the other components in the room (refer to Section A.4 for additional details), a 'neck' equipped with two servos, thereby providing two degrees of freedom (yaw and pitch), and a ZED 2 camera mounted on top (see Section A.3 for further information about the camera). An NVIDIA Jetson Orin computer is also incorporated into the robot's body to control all the mentioned devices (refer to Section A.2 for more details about this computer). All these components are powered by a battery with the following specifications:

- 48V/52V 24AH (100-1800W)
- Nominal Voltage: 48V/52V
- Rated Capacity: 24AH
- Battery Cell: 3000mAH
- Max Constant Discharge Current: 50A(BMS)
- Charger: 54.6V/58.8V 4A
- Discharge Port: A pair of XT60

A.1.3 Mapping the Room, Localisation and ROS Navigation System

This section details the steps required to enable the ROS navigation stack on the RB-1 base. The process commences with recording and saving a map of the room. To initiate map recording, open a terminal and execute the following commands:

```
$ ROS_NAMESPACE=robot roslaunch rb1_base_localization gmapping.launch
$ roslaunch rviz rviz
```

The first command starts the *gmapping* server, which begins map recording, while the second command opens *RViz*, a visual tool in ROS that displays the map as it is generated (ensure the correct *RViz* configuration is used). The robot is manually navigated across the entire space to be mapped. After recording the full map of the room, a similar result to that shown in Figure A.3 should be achieved.

To store the recorded map as a file, execute the following command, which saves the map to the home directory by default:

```
$ ROS_NAMESPACE=robot roslaunch rb1_base_localization map_saver.launch
```

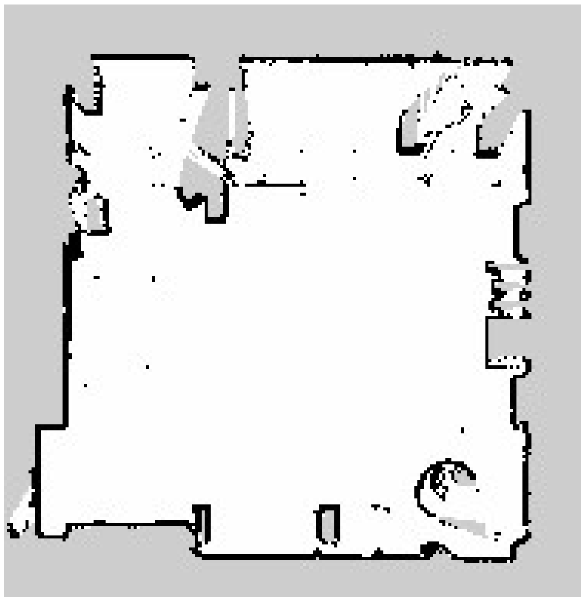
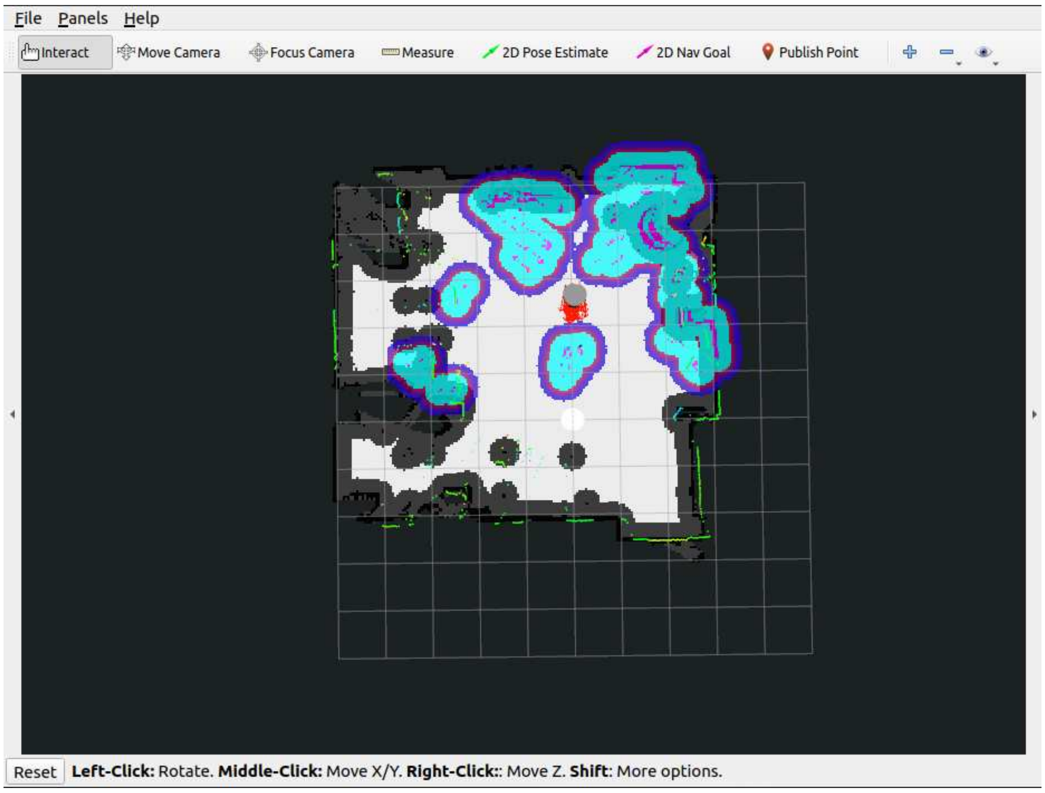


Figure A.3: Map recorded of the ARP Laboratory space using the laser sensor of the RB-1 base.



Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click::** Move Z. **Shift:** More options.

Figure A.4: Screenshot of *RViz* tool showing the global and local costmaps of TEB planner, the position of the robot, the ACML particles in red and the tools ribbon on the top.

Now that the map is available, the Adaptive Monte Carlo Localisation (AMCL) [196] module can be utilised to localise the robot on the map. To start, the first step is to load the recently recorded map with the following command:

```
$ ROS_NAMESPACE=robot roslaunch rb1_base_localization map_server.launch
```

Once the map is loaded and visible in *RViz*, the localisation program can be started by executing:

```
$ ROS_NAMESPACE=robot roslaunch rb1_base_localization amcl.launch
```

A multitude of red arrows will appear in *RViz*. Select the tool in *RViz* to indicate the robot's estimated position (the tool is highlighted with a green arrow in Figure A.4), then move the robot around the room until these red arrows nearly vanish, as depicted in Figure A.4. After this step, the robot will be localized within the room. The final step is to initialize the navigation system, which uses the Timed Elastic Band (TEB) planner [154]:

```
$ ROS_NAMESPACE=robot roslaunch rb1_base_localization move_base.launch
```

With the navigation system operational, goals can be set for the robot on the map, either via *RViz* or by publishing a ROS topic with the desired coordinates. The tool to set the goal can be seen in the capture of Figure A.4. The robot will navigate towards the set goal while adhering to the global and local cost maps of the TEB planner.

A.2 NVIDIA Jetson Orin

The Jetson Orin is a compact computer developed by NVIDIA, specifically aimed at AI applications. According to NVIDIA, this series of products are the world's most powerful AI computers designed for energy-efficient autonomous machines. NVIDIA Jetson Orin modules can deliver up to 275 trillion operations per second (TOPS), offering eight times the performance of the previous generation. This allows for multiple concurrent AI inference pipelines, along with high-speed interface support for multiple sensors.

In the ARP laboratory, the specific Jetson Orin module used is the Jetson AGX Orin Developer Kit, depicted in Figure A.5. More detailed information regarding the specifications of this computer can be found on the NVIDIA website: `https://www.nvidia.com/`

`en-gb/autonomous-machines/embedded-systems/jetson-orin/`

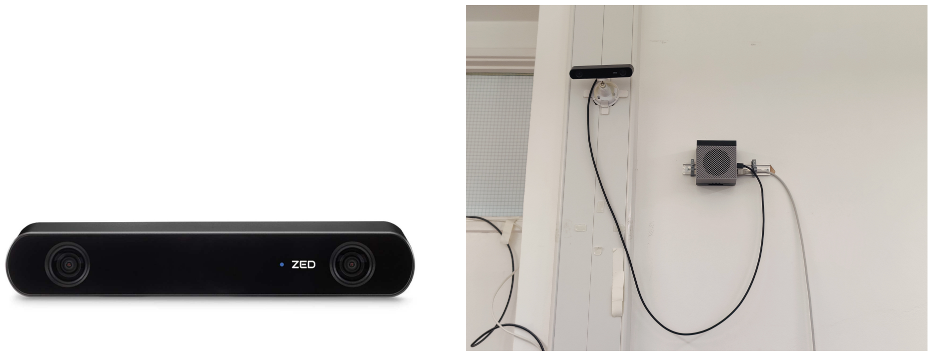
Figure A.5: Jetson AGX Orin Developer Kit used in the ARP laboratory.

A.3 Cameras

There are a total of five cameras installed in the ARP laboratory, and they are of two different types: three Intel RealSense depth cameras D455 and two ZED2 cameras from Stereolabs. One of the ZED 2 cameras is mounted on top of the robot, as mentioned in Section A.1.2, and the other one, along with the three D455 cameras, is placed on the walls of the laboratory. Figures A.6b and A.7b show examples of the placement on the wall for each of the camera types. Each camera is connected to an NVIDIA Jetson Orin for image capture and processing. The characteristics of both camera models are detailed in the following sections.

A.3.1 ZED 2 Camera

As stated by the manufacturer, the ZED 2 is the first stereo camera that employs neural networks to replicate human vision, with the ability to perceive depth. It comes equipped with a built-in IMU, Barometer, and Magnetometer, enabling the capture of real-time synchronized inertial, elevation, and magnetic field data alongside image and depth. The ZED 2 features two 16:9 native sensors and ultra-sharp 8-element all-glass lenses that can capture video and depth with a field of view (FOV) of up to 120° and a resolution of up to 2.2K. These cameras benefit from a wide-angle FOV, an advanced sensor stack and thermal calibration for significantly improved positional tracking accuracy. For more detailed specifications, such as the frame rate for each resolution, please refer to the manufacturer's website: `https://www.stereolabs.com/zed-2/`.

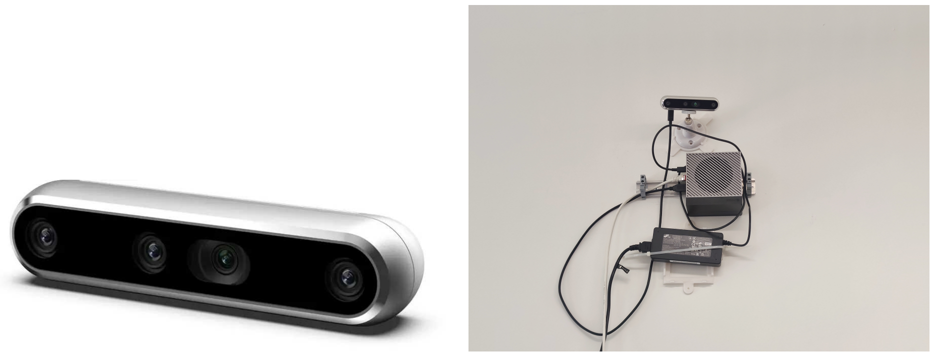


(a) **(b)**

Figure A.6: ZED 2 camera from Stereolabs (a) and its location on the wall (b)

A.3.2 Intel RealSense Depth Camera D455

The D455 extends the distance between the depth sensors to 95 mm, which reduces the depth error to less than 2% at 4 meters. To enhance the RGB image and the correspondence between the depth and RGB images, the RGB sensor includes a global shutter and matches the depth FOV. This camera also integrates an IMU, which helps refine depth awareness in any situation where the camera is moving, thereby enhancing environmental awareness for robotics and drones. Figure A.7 illustrates the camera and its location on one of the walls. More information about the specifications can be found at `https://www.intelrealsense.com/depth-camera-d455/`.



(a) **(b)**

Figure A.7: Intel RealSense Depth Camera D455 (a) and its location on the wall (b)

A.4 Communications

To facilitate communication among all the computers in the ARP laboratory, a local network was established using a switch and a high-speed access point, as depicted in Figure A.8, to ensure connectivity with the robot. Specifically, the access point used is the wAP 60Gx3 Access Point. More information on this can be found at `https://mikrotik.com/product/wap_60gx3_ap`.

The wAP 60Gx3 Access Point is a novel model designed for the 60 GHz spectrum. It is equipped with an antenna array that supports a wide angle of coverage, specifically optimized for multipoint operation. The 96 antenna elements work in conjunction with beamforming technology to provide connectivity for up to eight 60 GHz client devices simultaneously, within a 180-degree field of view. Because of its high operating frequency, it does not interfere with the bandwidth of other systems such as WiFi. Moreover, its reach is limited, providing coverage only within the room's area.

There is one wAP 60Gx3 antenna located on the wall and another one installed on the robot. The one on the wall is connected via Ethernet to the switch, which in turn is connected via Ethernet to all the computers linked to each of the cameras and to a central computer. This setup creates a local network that is not connected to the internet but interconnects all the elements in the ARP laboratory.

Figure A.8: wAP 60Gx3 Access Point

A.5 EPS Machine Learning Server

In this final section, the primary specifications of the Aston College of Engineering and Physical Sciences (EPS) Machine Learning Server are presented. This server has been employed extensively for training and testing the majority of deep learning models developed throughout this thesis. The specifications are as follows:

- 10x NVIDIA RTX A6000 48GB GPUs (each: 10,752 CUDA cores, 336 3rd-gen Tensor cores, 84 2nd-gen RT cores)
- 2x Intel Xeon Gold 5218R - 2.1GHz - 20 cores/40 threads - 27.5MB cache.
- 768GB (12x64GB) 2933MHz DDR4 RAM.
- 28TB storage (7x 4TB Intel DC4510 NVME)

Appendix B

Graph Neural Network Over-Smooth the Output?

This appendix introduces a small-scale study designed to assess the effectiveness of the Graph Neural Network variants employed in this thesis when used with the similar data referenced in Chapters 4 and 5. As noted in Chapter 3, a constraint of deep GNNs is their tendency to produce over-smooth outputs as they increase in depth. The objective of this study is to ascertain if this issue impacted the outcomes of the cost maps produced in the chapters mentioned above. Given that these cost maps are 2D images derived from zero-dimensional data, no ground truth exists against which to validate the expected results. The generated cost maps, as illustrated, are largely isotropic and display smooth-edged blobs. This experiment aims to determine whether this kind of output is what should be anticipated when bootstrapping our zero-dimensional dataset, or if it results from the over-smoothing problem inherent in GNNs.

The experimental design entails utilizing a Graph Neural Network combined with a CNN with identical characteristics to those utilized in Chapter 5. In this case, we want to approximate non-linear (hard edges) and non-isotropic functions. In contrast to the work in Chapter 5, the labels are not provided by participants but are instead the pixel values of the functions depicted in Figure B.2. The graphs, scenarios, model architecture, and hyperparameter tuning approach employed in this study are analogous to those utilized

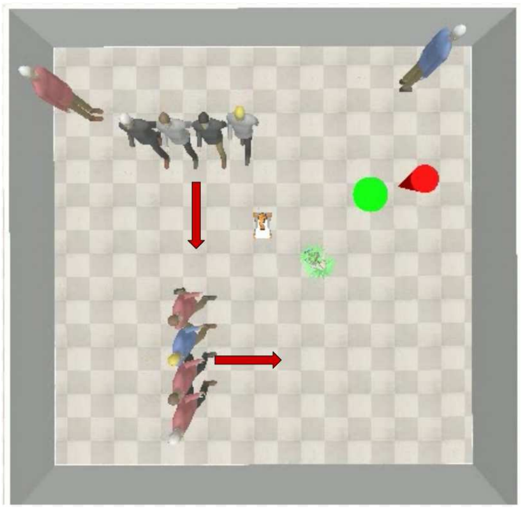
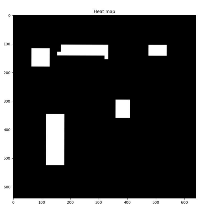


Figure B.1: A bird's-eye view of one of the scenarios used for testing the functions. The red arrows indicate in which direction the group of humans are moving. There are two static humans and a static object (plant).

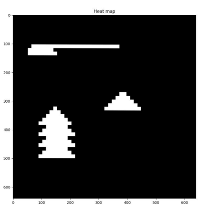
in Chapter 5, with the sole exception being the data used for prediction (i.e., the ground truth).

B.1 Dataset and Functions

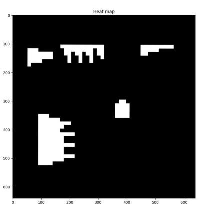
The datasets of graphs used in this experiment are identical to those of Chapter 5. They are created from the same set of short videos. Figure B.1, shows an example of one video frame where we have a group of dynamic humans, static humans and an object.



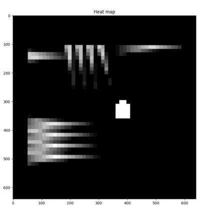
(a) Function 1



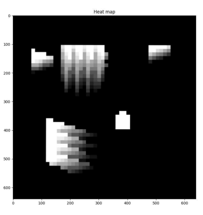
(b) Function 2



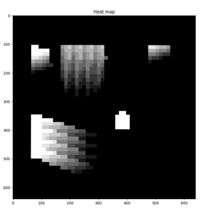
(c) Function 3



(d) Function 4



(e) Function 5



(f) Function 6

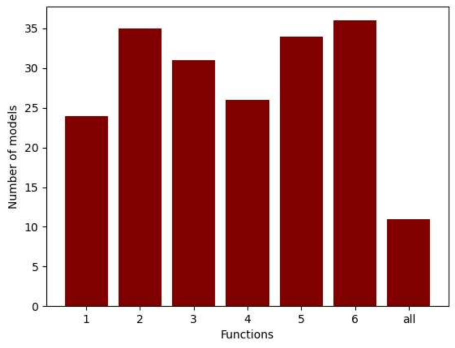
Figure B.2: Examples of graphs

The different functions are described as follows:

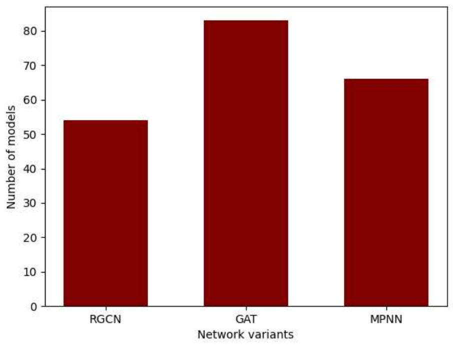
- Function 1 (Figure B.2a): Each human and object in the scenario is represented as a squared oriented in the same direction.
- Function 2 (Figure B.2b): Each entity in the room is represented as a non-oriented triangle.
- Function 3 (Figure B.2c): Same as function 2 but in this case the triangles follow the same orientation of the entities and the objects are represented by not-oriented squares.
- Function 4 (Figure B.2d): Slight longer triangles with a gradient of values having the largest values in the barycenter and lowest values in the edges. The superposition of several triangles is calculated with the sum of their values.
- Function 5 (Figure B.2e): Same as function 3 but in this case we take 3 frames separated by 1 second each. The triangles of the former frames have higher values than the later frames.
- Function 6 (Figure B.2f): Same as function 5 but adding several additional triangles per entity extrapolated from its trajectory.

B.2 Experimentation

Three different variants of GNNs of the ones explained in Chapter 3 were trained, namely, GAT, RGCN and MPNN. Each of these networks is trained in all the different functions, one at a time. The hyperparameter tuning method has been a random search, training a random GNN variant to predict a random function with random hyperparameters each time. A total of 156 models have been trained. Plots in Figure B.3 shows the models trained by function (B.3a) and by GNN variant (B.3b).



(a)



(b)

Figure B.3: Number of models trained by function (a) and by GNN model (b)

B.3 Results and Discussion

In this section, we can see the results of the losses with models with a different number of layers (see Figure B.4). As expected, there are no significant gains when making the network deeper.

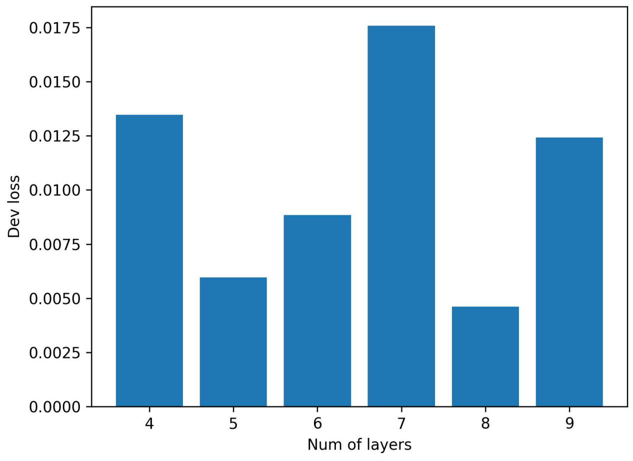
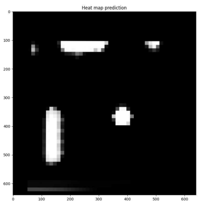


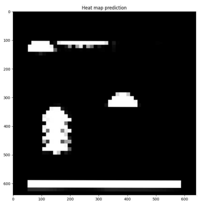
Figure B.4: Best losses per number of layers of the model for function 6. The results consider all the different GNN variants (average)

In Figure B.5, we can see the visual results of the best models for each of the functions. As you can see, the hard edges in all the functions are approximated correctly, and there is no sign of over-smoothing. In the results for function 4, we can see that the GNN can approximate soft and hard edges at the same time. It also captures the non-isotropies as shown in functions 2, 3 and 4 where the triangles are clearly recognisable.

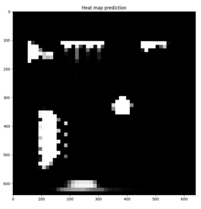
In conclusion, this small study proves that the GNN variants used in this thesis are



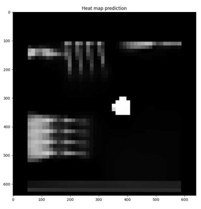
(a) Function 1



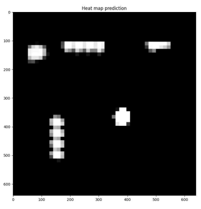
(b) Function 2



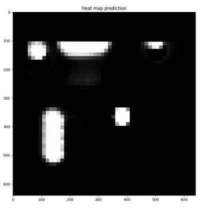
(c) Function 3



(d) Function 4



(e) Function 5



(f) Function 6

Figure B.5: Examples of graphs

expressive enough for solving the problems posed in Chapters 4 and 5. Besides, we can see that a few layers are enough in all cases and that deep does not help the overall network performance.

Appendix C

Calibration Process of the Cameras

The calibration of cameras in the laboratory is detailed in this appendix. The process determines the intrinsic parameters of all available cameras and the extrinsic parameters relative to the desired world frame of reference. For each camera a, the calibration matrices T_a, which transform world coordinates into camera coordinates, adhere to the form of Equation C.1.

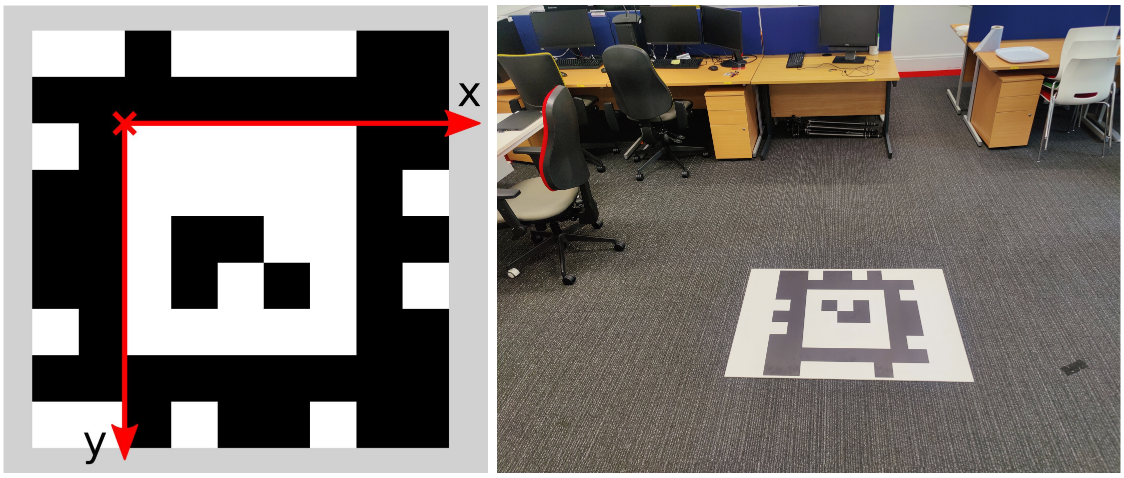
$$T_a = \begin{pmatrix} -f_{a,x} & 0 & c_{a,x} \\ 0 & -f_{a,y} & c_{a,y} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{a,3\times 3} & t_{a,3\times 1} \end{pmatrix} \tag{C.1}$$

Here, $f_{a,x}$ and $f_{a,y}$ are the horizontal and vertical focal lengths of camera a, while $c_{a,x}$ and $c_{a,y}$ are the pixel coordinates of the image centre. The rotation matrix and translation vector from the world to the camera frame of reference are $R_{a,3\times 3}$ and $t_{a,3\times 1}$, respectively. The intrinsic parameters are represented by the first matrix in Equation C.1, and the second matrix corresponds to the extrinsics.

This appendix is divided into two sections. Section C.1 outlines how the transformation matrix is calculated for each of the cameras from a common frame of reference, while Section C.2 illustrates how to manually calculate the intrinsics and presents a comparison with the intrinsics provided by the cameras' manufacturer's API.

C.1 Calculation of the Transformation Matrices.

To calibrate the extrinsics of all the cameras in the room using the same frame of reference, an AprilTag, similar to the one displayed in Figure C.1a, is employed. The AprilTag is placed on the room floor where it is visible from all the wall-mounted cameras, as shown in Figure C.1b. The intrinsic parameters, including focal lengths and centre points, can be calculated via a call to the API of each camera.



(a) **(b)**

Figure C.1: The variant of AprilTag used for the extrinsic calibrations of the cameras with the direction of the axis (Fig. C.1a) and its placement on the floor of the room (Fig. C.1b).

Once the cameras are fixed and the AprilTag is positioned, taking into account the desired direction of the frame of reference axis, a script is executed on each camera simultaneously. This script transmits the captured images and intrinsic parameters of each camera over the network. A central computer receives these images and intrinsic parameters and uses a third-party library[1] to calculate the extrinsics for each camera.

The extrinsics are integrated with the intrinsics to produce the final transformation matrices following the form of Equation C.1. These matrices are stored in a pickle file for future usage.

To verify the accuracy of the calibration script, a small experiment is conducted that involves calculating the distance from each camera to the centre of the AprilTag using the transformation matrices. For each camera a, the vector $(0, 0, 0, 1)$ is multiplied by its

[1]The library used to obtain the extrinsic using AprilTags is called *dt-apriltags* and the code is available at: `https://github.com/duckietown/lib-dt-apriltags`

respective transformation matrix T_a, which yields the homogeneous 3D coordinates of the camera's position with respect to the world frame of reference, $\vec{p_a}$. This calculation is represented in the following equation:

$$\vec{p_a} = T_a \times (0, 0, 0, 1) \tag{C.2}$$

With the position $\vec{p_a}$ of each camera determined, the Euclidean distance to the AprilTag can be calculated. Once these distances are computed, a laser meter is used to manually measure the distance from the AprilTag to each camera. Table C.1 shows the results for both types of measurements. As can be observed, the distances are nearly identical, confirming the precision of the calibration process.

Cameras	Distances to the origin (meters)	
	Manual	Calibration
A	4.660	4.650
B	5.824	5.736
C	4.553	4.496
D	5.840	5.820

Table C.1: Results of the manual testing of the calibration.

C.2 Manual Calculation of the Intrinsics Parameters and Comparative with the Values Provided by the API of the Cameras.

As mentioned in the previous section, the intrinsic parameters of each camera can be easily obtained from their respective APIs. This section outlines how these parameters can be manually calculated and provides a comparison with the manufacturer-provided parameters to evaluate their precision.

For the calculation of the intrinsic parameters, a checkerboard pattern, such as the one shown in Figure C.2a, is used. For each camera, a script that utilizes the *OpenCV* library is run to determine the intrinsic parameters of the camera when the checkerboard pattern is presented, as shown in Figure C.2b. Specifically, the functions used for the calibration are *findChessboardCorners()* to locate the checkerboard and *calibrateCamera()* to obtain the intrinsic parameters. Additional information about these functions can be found at `https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html`.

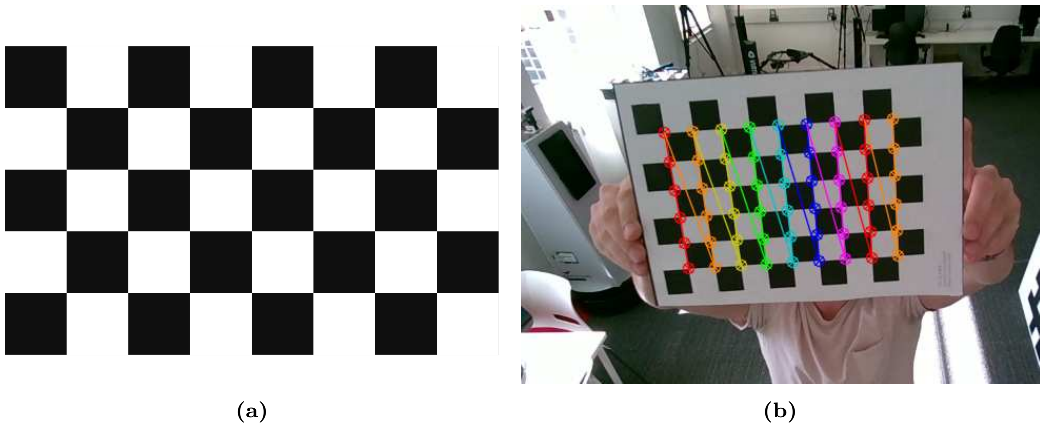


Figure C.2: Checkerboard pattern used for the intrinsic calibration of the cameras (Fig. C.2a) and the output during the calibration process by *OpenCV* (Fig. C.2b).

The results of the manual intrinsic calibration for each camera are presented in Table C.2, while the values provided by the cameras' APIs are displayed in Table C.3. Both calibrations use a resolution of 848×480. As can be observed, the values from both tables are very similar, which corroborates the accuracy of the factory parameters of the cameras.

Cameras	fx	fy	Cx	Cy
tracker a	423.475	423.884	420.693	237.982
tracker b	416.305	416.983	424.746	237.451
tracker c	423.386	424.121	424.189	246.160
tracker d	425.946	426.205	423.792	241.345

Table C.2: Manual calibration results for resolution 848x480

Cameras	fx	fy	Cx	Cy
tracker a	420.736	420.424	418.543	236.891
tracker b	420.839	420.434	421.197	239.158
tracker c	422.513	422.243	422.971	246.832
tracker d	422.103	421.955	422.904	241.955

Table C.3: Factory calibration values for resolution 848x480

List of References

[1] W. Abdulla. Mask R-CNN for object detection and instance segmentation on Kerasand TensorFlow. `https://github.com/matterport/Mask_RCNN`, 2017. GitHub repository.

[2] G. Adaimi, S. Kreiss, and A. Alahi. Perceiving traffic from aerial images. *ArXiv*, abs/2009.07611, 2020.

[3] J. K. Aggarwal and L. Xia. Human activity recognition from 3d data: A review. *Pattern Recognition Letters*, 48:70–80, 2014.

[4] A. Agrawal and R. Paulus. Intelligent traffic light design and control in smart cities: A survey on techniques and methodologies. *International Journal of Vehicle Information and Communication Systems*, 5(4):436–481, 2020. ISSN 17418208. doi: 10.1504/IJVICS.2020.111456.

[5] A. Agrawal, R. Arora, A. Datta, S. Banerjee, B. Bhowmick, K. M. Jatavallabhula, M. Sridharan, and M. Krishna. Clipgraphs: Multimodal graph networks to infer object-room affinities, 2023.

[6] E. Ahmed, I. Yaqoob, A. Gani, M. Imran, and M. Guizani. Internet-of-things-based smart environments: state of the art, taxonomy, and open research challenges. *IEEE Wireless Communications*, 23(5):10–16, 2016.

[7] U. Alon and E. Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

[8] S. Amin, M. Andriluka, M. Rohrbach, and B. Schiele. Multi-view pictorial structures for 3d human pose estimation. In *Bmvc*, volume 1, 2013.

[9] R. B. Arantes, G. Vogiatzis, and D. R. Faria. Csc-gan: Cycle and semantic consistency for dataset augmentation. In *Advances in Visual Computing: 15th International Symposium, ISVC 2020, San Diego, CA, USA, October 5–7, 2020, Proceedings, Part I 15*, pages 170–181. Springer, 2020.

[10] P. Bachiller, **Rodriguez-Criado, Daniel**, R. R. Jorvekar, P. Bustos, D. R. Faria, and L. J. Manso. A graph neural network to model disruption in human-aware robot navigation. *Multimedia Tools and Applications*, pages 1–19, 2021.

[11] R. Baghel, A. Kapoor, P. Bachiller, R. R. Jorvekar, **Rodriguez-Criado, Daniel**, and L. J. Manso. A toolkit to generate social navigation datasets. In *Workshop of Physical Agents*, pages 180–193. Springer, 2020.

[12] R. Bajpai and D. Joshi. Movenet: A deep neural network for joint profile prediction across variable walking speeds and slopes. *IEEE Transactions on Instrumentation and Measurement*, 70:1–11, 2021. doi: 10.1109/TIM.2021.3073720.

[13] E. Barmpounakis, G. M. Sauvin, and N. Geroliminis. Lane Detection and Lane-Changing Identification with High-Resolution Data from a Swarm of Drones. *Transportation Research Record*, 2674(7):1–15, 2020. ISSN 21694052. doi: 10.1177/0361198120920627.

[14] P. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, 2016.

[15] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv: 1806.01261*, pages 1–40, 2018. ISSN 0031-1820. URL `http://arxiv.org/abs/1806.01261`.

[16] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation.* ThinkMind, 2011.

[17] V. Belagiannis, S. Amin, M. Andriluka, B. Schiele, N. Navab, and S. Ilic. 3d pictorial structures for multiple human pose estimation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1669–1676, 2014. doi: 10.1109/CVPR.2014.216.

[18] V. Belagiannis, S. Amin, M. Andriluka, B. Schiele, N. Navab, and S. Ilic. 3D Pictorial Structures Revisited: Multiple Human Pose Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):1929–1942, 2016. ISSN 01628828. doi: 10.1109/TPAMI.2015.2509986.

[19] Y. Bengio. Practical recommendations for gradient-based training of deep architectures, 2012. URL `https://arxiv.org/abs/1206.5533`.

[20] S. Biswas, S. Sinha, K. Gupta, and B. Bhowmick. Lifting 2d human pose to 3d: A weakly supervised approach. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2019.

[21] H. L. Bradwell, K. J. Edwards, R. Winnington, S. Thill, and R. B. Jones. Companion robots for older people: importance of user-centred design demonstrated through observations and focus groups comparing preferences of older people and roboticists in south west england. *BMJ Open*, 9(9), 2019. ISSN 2044-6055. doi: 10.1136/bmjopen-2019-032468. URL `https://bmjopen.bmj.com/content/9/9/e032468`.

[22] L. Bridgeman, M. Volino, J.-Y. Guillemaut, and A. Hilton. Multi-person 3d pose estimation and tracking in sports. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2487–2496, 2019. doi: 10.1109/CVPRW.2019.00304.

[23] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[24] M. Camplani, A. Paiement, M. Mirmehdi, D. Damen, S. Hannuna, T. Burghardt, and L. Tao. Multiple human tracking in rgb-depth data: a survey. *IET Computer Vision*, 11(4):265–285, 2017. doi: https://doi.org/10.1049/iet-cvi.2016.0178. URL `https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cvi.2016.0178`.

[25] W. Cao, Y. Lu, and Z. He. Geometric Algebra Representation and Ensemble Action Classification Method for 3D Skeleton Orientation Data. *IEEE Access*, 7:132049–132056, 2019. ISSN 21693536. doi: 10.1109/ACCESS.2019.2940291.

[26] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[27] F. Cardinaux, D. Bhowmik, C. Abhayaratne, and M. Hawley. Video based technology for ambient assisted living: A review of the literature. *JAISE*, 3:253–269, 05 2011. doi: 10.3233/AIS-2011-0110.

[28] K. Charalampous, I. Kostavelis, and A. Gasteratos. *Recent trends in social aware robot navigation: A survey*, volume 93, pages 85–104. Elsevier B.V., 2017.

[29] C. Chen, S. Hu, P. Nikdel, G. Mori, and M. Savva. Relational graph learning for crowd navigation. *arXiv preprint arXiv:1909.13165*, 2019.

[30] C. Chen, Y. Liu, S. Kreiss, and A. Alahi. Crowd-Robot Interaction: Crowd-aware Robot Navigation with Attention-based Deep Reinforcement Learning. In *International Conference on Robotics and Automation (ICRA)*, pages 6015–6022. IEEE, 2019. URL `http://arxiv.org/abs/1809.08835`.

[31] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020.

[32] T. Chen, K. Zhou, K. Duan, W. Zheng, P. Wang, X. Hu, and Z. Wang. Bag of Tricks for Training Deeper Graph Neural Networks: A Comprehensive Benchmark Study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):2769–2781, 2022. ISSN 19393539. doi: 10.1109/TPAMI.2022.3174515.

[33] T. L. Chen, M. Ciocarlie, S. Cousins, P. Grice, K. Hawkins, K. Hsiao, C. Kemp, C. H. King, D. Lazewatsky, A. E. Leeper, H. Nguyen, A. Paepcke, C. Pantofaru, W. Smart, and L. Takayama. Robots for humanity: Using assistive robotics to empower people with disabilities. *IEEE Robotics and Automation Magazine*, 20(1):30–39, 2013. ISSN 10709932. doi: 10.1109/MRA.2012.2229950.

[34] T. L. Chen, M. Ciocarlie, S. Cousins, P. Grice, K. Hawkins, K. Hsiao, C. Kemp, C. H. King, D. Lazewatsky, A. E. Leeper, H. Nguyen, A. Paepcke, C. Pantofaru, W. Smart, and L. Takayama. Robots for humanity: Using assistive robotics to empower people with disabilities. *IEEE Robotics and Automation Magazine*, 20(1):30–39, 2013. ISSN 10709932.

[35] Y. Chen, C. Liu, B. E. Shi, and M. Liu. Robot Navigation in Crowds by Graph Convolutional Networks with Attention Learned from Human Gaze. *IEEE Robotics and Automation Letters*, 5(2):2754–2761, 2020. ISSN 23773766. doi: 10.1109/LRA.2020.2972868.

[36] Y. Chen, C. Liu, B. E. Shi, and M. Liu. Robot navigation in crowds by graph convolutional networks with attention learned from human gaze. *IEEE Robotics and Automation Letters*, 5(2):2754–2761, 2020.

[37] Y. F. Chen, M. Everett, M. Liu, and J. P. How. Socially aware motion planning with deep reinforcement learning. *IEEE International Conference on Intelligent Robots and Systems*, 2017-Septe:1343–1350, 2017. ISSN 21530866.

[38] Z. Chen, L. Chen, S. Villar, and J. Bruna. Can graph neural networks count substructures? *Advances in Neural Information Processing Systems*, 2020-December (NeurIPS), 2020. ISSN 10495258.

[39] J. Choi, B.-J. Lee, and B.-T. Zhang. Human body orientation estimation using convolutional neural network. *arXiv preprint arXiv:1609.01984*, 2016.

[40] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[41] J. Cohen. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin*, 70(4):213, 1968.

[42] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Velickovic. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 2020-Decem(NeurIPS), 2020. ISSN 10495258.

[43] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, dec 1989. ISSN 1435-568X. doi: 10.1007/BF02551274. URL `http://link.springer.com/10.1007/BF02551274`.

[44] D. N. Thang et al. Deep Learning-based Multiple Objects Detection and Tracking System for Socially Aware Mobile Robot Navigation Framework. *NICS 2018 - Proceedings of 2018 5th NAFOSTED Conference on Information and Computer Science*, pages 436–441, 2019. doi: 10.1109/NICS.2018.8606878.

[45] C. Dondrup and M. Hanheide. Qualitative constraints for human-aware robot navigation using velocity costmaps. In *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 586–592. IEEE, 2016.

[46] J. Dong, Q. Fang, W. Jiang, Y. Yang, Q. Huang, H. Bao, and X. Zhou. Fast and robust multi-person 3d pose estimation and tracking from multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[47] D. Drover, R. M. V, C.-H. Chen, A. Agrawal, A. Tyagi, and C. P. Huynh. Can 3d pose be learned from 2d projections alone? In L. Leal-Taixé and S. Roth, editors, *Computer Vision – ECCV 2018 Workshops*, pages 78–94, Cham, 2019. Springer International Publishing. ISBN 978-3-030-11018-5.

[48] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, 2015.

[49] F. Fang, M. Shi, K. Qian, B. Zhou, and Y. Gan. A human-aware navigation method for social robot based on multi-layer cost map. *International Journal of Intelligent Robotics and Applications*, 4:308–318, 2020.

[50] J. Fernández, J. M. Cañas, V. Fernández, and S. Paniego. Robust Real-Time Traffic Surveillance with Deep Learning. *Computational Intelligence and Neuroscience*, 2021, 2021. ISSN 16875273. doi: 10.1155/2021/4632353.

[51] G. Ferrer and A. Sanfeliu. Proactive kinodynamic planning using the Extended Social

Force Model and human motion prediction in urban environments. *IEEE International Conference on Intelligent Robots and Systems*, pages 1730–1735, 2014. ISSN 21530866.

[52] K. Fukushima and S. Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[53] D. Garg, M. Chli, and G. Vogiatzis. Fully-Autonomous, Vision-based Traffic Signal Control: from Simulation to Reality. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 1:454–462, 2022. ISSN 15582914.

[54] D. Gerónimo, A. M. López, A. D. Sappa, and T. Graf. Survey of pedestrian detection for advanced driver assistance systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(7):1239–1258, 2010. doi: 10.1109/TPAMI.2009.122.

[55] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[56] G. Glonek and A. Wojciechowski. Hybrid orientation based human limbs motion tracking method. *Sensors*, 17(12), 2017. ISSN 14248220. doi: 10.3390/s17122857.

[57] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `https://proceedings.mlr.press/v9/glorot10a.html`.

[58] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL `https://proceedings.mlr.press/v15/glorot11a.html`.

[59] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[60] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[61] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[62] C. C. Gordon, C. L. Blackwell, B. Bradtmiller, J. L. Parham, P. Barrientos, S. P. Paquette, B. D. Corner, J. M. Carson, J. C. Venezia, B. M. Rockwell, et al. 2012 anthropometric survey of us army personnel: Methods and summary statistics. Technical report, Army Natick Soldier Research Development and Engineering Center MA, 2014.

[63] M. Gori, G. Monfardini, and F. Scarselli. A new Model for Learning in Graph domains. *Proceedings of the International Joint Conference on Neural Networks*, 2:729–734, 2005.

[64] H. M. Gross, A. Scheidig, S. Müller, B. Schütz, C. Fricke, and S. Meyer. Living with a mobile companion robot in your own apartment - Final implementation and results of a 20-weeks field study with 20 seniors. *Proceedings - IEEE International Conference on Robotics and Automation*, 2019-May:2253–2259, 2019. ISSN 10504729.

[65] S. Haddad and S. K. Lam. Self-growing spatial graph networks for pedestrian trajectory prediction. *Proceedings - 2020 IEEE Winter Conference on Applications of Computer Vision, WACV 2020*, pages 1140–1148, 2020. doi: 10.1109/WACV45572.2020.9093456.

[66] E. T. Hall. *The hidden Dimension: Man's Use of Space in Public and Private*. The Bodley Head Ltd, London, UK, 1966.

[67] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[68] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.

[69] D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, 1995. ISSN 1063651X.

[70] S. Hemachandra, M. R. Walter, S. Tellex, and S. Teller. Learning spatial-semantic representations from natural language descriptions and scene classifications. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2623–2630. IEEE, 2014.

[71] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[72] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[73] H. Hu, J. Gu, Z. Zhang, J. Dai, and Y. Wei. Relation Networks for Object Detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018. ISBN 9781538664209.

[74] W. Hu, C. Zhang, F. Zhan, L. Zhang, and T.-T. Wong. Conditional directed graph convolution for 3d human pose estimation. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 602–611, 2021.

[75] N. T. Huang and S. Villar. A short tutorial on the weisfeiler-lehman test and its variants. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, jun 2021. doi: 10.1109/icassp39728.2021.9413523. URL `https://doi.org/10.1109%2Ficassp39728.2021.9413523`.

[76] W. Huang, Y. Rong, T. Xu, F. Sun, and J. Huang. Tackling over-smoothing for general graph convolutional networks. *arXiv preprint arXiv:2008.09864*, 2020.

[77] Y. Huang, H. Bi, Z. Li, T. Mao, and Z. Wang. STGAT: Modeling spatial-temporal interactions for human trajectory prediction. *Proceedings of the IEEE International*

Conference on Computer Vision, 2019-October:6271–6280, 2019. ISSN 15505499. doi: 10.1109/ICCV.2019.00637.

[78] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985. URL `http://scholar.google.de/scholar.bib?q=info:IkrWWF2JxwoJ:scholar.google.com/&output=citation&hl=de&ct=citation&cd=0`.

[79] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[80] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339, 7 2014.

[81] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[82] A. Jain, J. Tompson, M. Andriluka, G. W. Taylor, and C. Bregler. Learning human pose estimation features with convolutional networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, pages 1–11, 2014.

[83] S. James, M. Freese, and A. J. Davison. Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*, 2019.

[84] H. Joo, H. Liu, L. Tan, L. Gui, B. C. Nabbe, I. A. Matthews, T. Kanade, S. Nobuhara, and Y. Sheikh. Panoptic studio: A massively multiview system for social motion capture. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 3334–3342. IEEE Computer Society, 2015.

[85] C. Joshi. Transformers are graph neural networks. *The Gradient*, 2020.

[86] T. Kailath. The divergence and Bhattacharyya distance measures in signal selection. *IEEE transactions on communication technology*, 15(1):52–60, 1967.

[87] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. URL `https://arxiv.org/abs/1412.6980`.

[88] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[89] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2022.

[90] T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv:1609.02907*, pages 1–14, 2016. URL `http://arxiv.org/abs/1609.02907`.

[91] R. Kirby, R. Simmons, and J. Forlizzi. Companion: A constraint-optimizing method for person-acceptable navigation. In *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 607–612, 2009.

[92] M. Kocabas, S. Karagoz, and E. Akbas. Self-supervised learning of 3d human pose using multi-view geometry. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1077–1086, 2019.

[93] M. Kollmitz, K. Hsiao, J. Gaa, and W. Burgard. Time dependent planning on a layered social cost map for human-aware robot navigation. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–6, 2015. doi: 10.1109/ECMR.2015.7324184.

[94] I. Kostavelis, K. Charalampous, A. Gasteratos, and J. K. Tsotsos. Robot navigation via spatial and temporal coherent semantic maps. *Engineering Applications of Artificial Intelligence*, 48:173–187, 2016.

[95] S. Kreiss, L. Bertoni, and A. Alahi. Pifpaf: Composite fields for human pose estimation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11969–11978, Los Alamitos, CA, USA, jun 2019. IEEE Computer Society. doi: 10.1109/CVPR.2019.01225. URL `https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.01225`.

[96] S. Kreiss, L. Bertoni, and A. Alahi. Pifpaf: Composite fields for human pose estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[97] S. Kreiss, L. Bertoni, and A. Alahi. Openpifpaf: Composite fields for semantic keypoint detection and spatio-temporal association. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[98] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[99] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, 2013. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2013.05.007. URL `http://www.sciencedirect.com/science/article/pii/S0921889013001048`.

[100] J. N. Kundu, S. Seth, V. Jampani, M. Rakesh, R. V. Babu, and A. Chakraborty. Self-supervised 3d human pose estimation via part guided novel image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6152–6162, 2020.

[101] S. Laible and A. Zell. Building local terrain maps using spatio-temporal classification for semantic robot localization. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4591–4597. IEEE, 2014.

[102] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1), 1977.

[103] P. Langley, B. Meadows, M. Sridharan, and D. Choi. Explainable agency for intelligent autonomous systems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(2):4762–4763, Feb. 2017. doi: 10.1609/aaai.v31i2.19108. URL `https://ojs.aaai.org/index.php/AAAI/article/view/19108`.

[104] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, 5:293–308, 2001.

[105] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and

L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[106] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient Back-Prop BT - Neural Networks: Tricks of the Trade. *Neural Networks: Tricks of the Trade*, 7700(Chapter 3):9–48, 2012. URL `http://link.springer.com/chapter/10.1007/978-3-642-35289-8_3/fulltext.html%5Cnpapers3://publication/doi/10.1007/978-3-642-35289-8_3`.

[107] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[108] B. Lewandowski, D. Seichter, T. Wengefeld, L. Pfennig, H. Drumm, and H.-M. Gross. Deep orientation: Fast and Robust Upper Body orientation Estimation for Mobile Robotic Applications. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 441–448, 2020. ISBN 9781728140049. doi: 10.1109/iros40897.2019.8968506.

[109] G. Li, M. Muller, A. Thabet, and B. Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9267–9276, 2019.

[110] Q. Li, F. Gama, A. Ribeiro, and A. Prorok. Graph neural networks for decentralized multi-robot path planning, 2019.

[111] S. Li and A. B. Chan. 3d human pose estimation from monocular images with deep convolutional neural network. In *Asian Conference on Computer Vision*, pages 332–347. Springer, 2014.

[112] S. Li, L. Da Xu, and S. Zhao. 5g internet of things: A survey. *Journal of Industrial Information Integration*, 10:1–9, 2018.

[113] J. Lin and G. H. Lee. Multi-view multi-person 3d pose estimation with plane sweep stereo. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11886–11895, 2021.

[114] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In D. Fleet, T. Pajdla,

B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10602-1.

[115] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11461–11471, 2022.

[116] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013.

[117] R. Maini and H. Aggarwal. A comprehensive review of image enhancement techniques. *CoRR*, abs/1003.4053, 2010. URL `http://arxiv.org/abs/1003.4053`.

[118] L. J. Manso, R. R. Jorvekar, D. R. Faria, P. Bustos, and P. Bachiller. Graph Neural Networks for Human-aware Social Navigation. *arXiv preprint arXiv:1909.09003*, 2019.

[119] L. J. Manso, P. Nunez, L. V. Calderita, D. R. Faria, and P. Bachiller. SocNav1: A Dataset to Benchmark and Learn Social Navigation Conventions. *arXiv e-prints*, art. arXiv:1909.02993, Sep 2019.

[120] A. Mateus, D. Ribeiro, P. Miraldo, and J. C. Nascimento. Efficient and robust pedestrian detection using deep learning for human-aware navigation. *Robotics and Autonomous Systems*, 113:23–37, 2019. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2018.12.007. URL `https://www.sciencedirect.com/science/article/pii/S0921889017306784`.

[121] D. Mehta, O. Sotnychenko, F. Mueller, W. Xu, M. Elgharib, P. Fua, H.-P. Seidel, H. Rhodin, G. Pons-Moll, and C. Theobalt. Xnect: Real-time multi-person 3d motion capture with a single rgb camera. *Acm Transactions On Graphics (TOG)*, 39(4):82–1, 2020.

[122] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[123] R. Möller, A. Furnari, S. Battiato, A. Härmä, and G. M. Farinella. A survey on human-aware robot navigation. *Robotics and Autonomous Systems*, 145:103837, 2021.

[124] F. Moreno-Noguer. 3d human pose estimation from a single image via distance matrix regression. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2823–2832, 2017.

[125] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

[126] K. D. B. Mudavathu, M. C. S. Rao, and K. Ramana. Auxiliary conditional generative adversarial networks for image data set augmentation. In *2018 3rd International Conference on Inventive Computation Technologies (ICICT)*, pages 263–269. IEEE, 2018.

[127] R. Nakano. A discussion of 'adversarial examples are not bugs, they are features': Adversarially robust neural style transfer. *Distill*, 2019. doi: 10.23915/distill.00019.4. https://distill.pub/2019/advex-bugs-discussion/response-4.

[128] M. Neunert, C. De Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli. Fast nonlinear Model Predictive Control for unified trajectory optimization and tracking. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1398–1404, 2016. ISBN 9781467380263. doi: 10.1109/ICRA.2016.7487274.

[129] A. Ng. Preventing "overfitting" of cross-validation data. *Proceedings of the Fourteenth International Conference on Machine Learning*, 01 1998.

[130] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[131] N. J. Nilsson. *Principles of artificial intelligence.* Morgan Kaufmann, 2014.

[132] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[133] K. Oono and T. Suzuki. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. *arXiv preprint arXiv:1905.10947*, pages 1–37, 2019. URL `http://arxiv.org/abs/1905.10947`.

[134] J. Oppenlaender. The creativity of text-to-image generation. In *Proceedings of the 25th International Academic Mindtrek Conference*, pages 192–202, 2022.

[135] E. Pacchierotti, H. I. Christensen, and P. Jensfelt. Human-robot embodied interaction in hallway settings: A pilot user study. In *IEEE International Workshop on Robot and Human Interactive Communication*, volume 2005, pages 164–171. IEEE, 2005. ISBN 0780392752. doi: 10.1109/ROMAN.2005.1513774.

[136] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[137] P. Patompak, S. Jeong, I. Nilkhamhang, and N. Y. Chong. Learning Proxemics for Personalized Human-Robot Social Interaction. *International Journal of Social Robotics*, 2019. ISSN 18754805.

[138] G. Pavlakos, X. Zhou, K. G. Derpanis, and K. Daniilidis. Coarse-to-fine volumetric prediction for single-image 3d human pose. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1263–1272, Los Alamitos, CA, USA, jul 2017. IEEE Computer Society. doi: 10.1109/CVPR.2017.139. URL `https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.139`.

[139] N. Pérez-Higueras, F. Caballero, and L. Merino. Learning Human-Aware Path Planning with Fully Convolutional Networks. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5897–5902, 2018. ISSN 10504729.

[140] S. Qi, W. Wang, B. Jia, J. Shen, and S. C. Zhu. Learning human-object interactions by graph parsing neural networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11213 LNCS:407–423, 2018. ISSN 16113349. doi: 10.1007/978-3-030-01240-3_25.

[141] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[142] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[143] H. Rhodin, J. Spörri, I. Katircioglu, V. Constantin, F. Meyer, E. Müller, M. Salzmann, and P. Fua. Learning monocular 3d human pose estimation from multi-view images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8437–8446, 2018.

[144] J. Rios-Martinez, A. Spalanzani, and C. Laugier. From Proxemics Theory to Socially-Aware Navigation: A Survey. *International Journal of Social Robotics*, 7(2):137–153, 2015. ISSN 18754805.

[145] **Rodriguez-Criado, Daniel**, P. Bachiller, P. Bustos, G. Vogiatzis, and L. J. Manso. Multi-camera torso pose estimation using graph neural networks. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 827–832. IEEE, 2020.

[146] **Rodriguez-Criado, Daniel**, P. Bachiller, and L. J. Manso. Generation of human-aware navigation maps using graph neural networks. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 19–32. Springer, Cham, 2021.

[147] **Rodriguez-Criado, Daniel**, M. Chli, G. Vogiatzis, and L. J. Manso. Synthesizing traffic datasets using graph neural networks. In *International Conference on Intelligent Transportation Systems*. IEEE, 2023.

[148] G. Rogez, P. Weinzaepfel, and C. Schmid. Lcr-net++: Multi-person 2d and 3d pose detection in natural images. *IEEE transactions on pattern analysis and machine intelligence*, 42(5):1146–1161, 2019.

[149] E. Rohmer, S. P. Singh, and M. Freese. Coppeliasim: A versatile and scalable robot simulation framework. In *Proc. Int. Conf. on Intelligent Robots and Systems*, pages 1321–1326, 2013.

[150] Y. Rong, W. Huang, T. Xu, and J. Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[151] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[152] A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 410–420, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `https://aclanthology.org/D07-1043`.

[153] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[154] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram. Trajectory modification considering dynamic constraints of autonomous robots. In *ROBOTIK 2012; 7th German Conference on Robotics*, pages 1–6. VDE, 2012.

[155] M. Rouse. What is artificial intelligence (AI)? `https://searchenterpriseai.techtarget.com/definition/AI-Artificial-Intelligence`, 2019. Online; accessed 9 January 2022.

[156] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control, 2018.

[157] F. Scarselli, M. Hagenbuchner, S. L. Yong, A. C. Tsoi, M. Gori, and M. Maggini. Graph neural networks for ranking web pages. *Proceedings - 2005 IEEE/WIC/ACM InternationalConference on Web Intelligence, WI 2005*, 2005:666–672, 2005.

[158] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling Relational Data with Graph Convolutional Networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10843 LNCS(1):593–607, 2018. ISSN 16113349.

[159] A. Sebti and H. Hassanpour. Body orientation estimation with the ensemble of logistic

regression classifiers. *Multimedia Tools and Applications*, 76(22):23589–23605, 2017. ISSN 15737721. doi: 10.1007/s11042-016-4129-0.

[160] N. Shafiee, T. Padir, and E. Elhamifar. Introvert: Human trajectory prediction via conditional 3d attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16815–16825, 2021.

[161] F. Shinmura, D. Deguchi, I. Ide, H. Murase, and H. Fujiyoshi. Estimation of human orientation using coaxial RGB-depth images. *VISAPP 2015 - 10th International Conference on Computer Vision Theory and Applications; VISIGRAPP, Proceedings*, 2:113–120, 2015. doi: 10.5220/0005305301130120.

[162] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[163] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[164] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[165] M. Sridharan and T. Mota. Combining commonsense reasoning and knowledge acquisition to guide deep learning in robotics, 2022.

[166] V. Srivastav, A. Gangi, and N. Padoy. Self-supervision on unlabelled or data for multi-person 2d/3d human pose estimation. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2020: 23rd International Conference, Lima, Peru, October 4–8, 2020, Proceedings, Part I 23*, pages 761–771. Springer, 2020.

[167] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

[168] K. Sun, B. Xiao, D. Liu, and J. Wang. Deep high-resolution representation learning for human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5693–5703, 2019.

[169] L. Sun, Z. Yan, S. M. Mellado, M. Hanheide, and T. Duckett. 3dof pedestrian trajectory prediction learned from long-term autonomous mobile robot deployment data. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5942–5948. IEEE, 2018.

[170] S. Sun, X. Zhao, Q. Li, and M. Tan. Inverse reinforcement learning-based time-dependent A* planner for human-aware robot navigation with local vision. *Advanced Robotics*, 34(13):888–901, 2020. ISSN 15685535.

[171] T. Taipalus and J. Ahtiainen. Human detection and tracking with knee-high mobile 2d lidar. In *2011 IEEE International Conference on Robotics and Biomimetics*, pages 1672–1677, 2011. doi: 10.1109/ROBIO.2011.6181529.

[172] S. Thomas and S. Passi. *PyTorch deep learning hands-on: apply modern AI techniques with CNNs, RNNs, GANs, reinforcement learning, and more.* Packt, 2019.

[173] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017. doi: 10.1109/IROS.2017.8202133.

[174] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper_files/paper/2014/file/e744f91c29ec99f0e662c9177946c627-Paper.pdf`.

[175] P. Trautman and A. Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 797–803. IEEE, 2010.

[176] X. Truong and T. D. Ngo. Toward socially aware robot navigation in dynamic and crowded environments: A proactive social motion model. *IEEE Transactions on Automation Science and Engineering*, 14(4):1743–1760, 2017.

[177] H. Tu, C. Wang, and W. Zeng. Voxelpose: Towards multi-camera 3d human pose estimation in wild environment. In *European Conference on Computer Vision*, pages 197–212. Springer, 2020.

[178] T. van der Heiden, C. Weiss, N. N. Shankar, and H. van Hoof. Social navigation with human empowerment driven reinforcement learning. In *arXiv preprint arXiv:2003.08158*, 2020. URL `http://arxiv.org/abs/2003.08158`.

[179] D. Vasquez, B. Okal, and K. O. Arras. Inverse Reinforcement Learning algorithms and features for robot navigation in crowds: An experimental comparison. *IEEE International Conference on Intelligent Robots and Systems*, pages 1341–1346, 2014. ISSN 21530866.

[180] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[181] A. Vega, L. J. Manso, D. G. Macharet, P. Bustos, and P. Núñez. Socially aware robot navigation system in human-populated and interactive environments based on an adaptive spatial density function and space affordances. *Pattern Recognition Letters*, 118:72–84, 2019. ISSN 01678655.

[182] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

[183] A. Vemula, K. Muelling, and J. Oh. Social attention: Modeling attention in human crowds. *arXiv*, pages 4601–4607, 2017. ISSN 23318422.

[184] R. Wahyu and A. Saputra. Human Body ' s Orientation Estimation Based On Depth Image. *2019 International Electronics Symposium (IES)*, pages 266–271, 2019.

[185] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*, 2022.

[186] J. Wang, S. Tan, X. Zhen, S. Xu, F. Zheng, Z. He, and L. Shao. Deep 3D human pose estimation: A review. *Computer Vision and Image Understanding*, 210(August

2020):103225, 2021. ISSN 1090235X. doi: 10.1016/j.cviu.2021.103225. URL `https://doi.org/10.1016/j.cviu.2021.103225`.

[187] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018.

[188] X. Wang. Intelligent multi-camera video surveillance: A review. *Pattern Recognit. Lett.*, 34:3–19, 2013.

[189] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[190] Z. Wang, Q. Lv, X. Lan, and Y. Zhang. Cross-lingual knowledge graph alignment via graph convolutional networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 349–357, 2018.

[191] B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series*, 2(9):12–16, 1968.

[192] J. Welsh. trt_pose. `https://github.com/NVIDIA-AI-IOT/trt_pose`, 2012. Accessed: 2022-06-09.

[193] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[194] S. Wu, S. Jin, W. Liu, L. Bai, C. Qian, D. Liu, and W. Ouyang. Graph-based 3d multi-person pose estimation using multi-view images. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11148–11157, 2021.

[195] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2020. ISSN 2162-237X.

[196] W. Xiaoyu, L. Caihong, S. Li, Z. Ning, and F. Hao. On adaptive monte carlo localization algorithm for the mobile robot based on ros. In *2018 37th Chinese Control Conference (CCC)*, pages 5207–5212, 2018. doi: 10.23919/ChiCC.2018.8482698.

[197] C. Xu, S. Chen, M. Li, and Y. Zhang. Invariant teacher and equivariant student for unsupervised 3d human pose estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3013–3021, 2021.

[198] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks?, 2018.

[199] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462. PMLR, 2018.

[200] Z. Yan, T. Duckett, and N. Bellotto. Online learning for human classification in 3d lidar-based tracking. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 864–871, 2017. doi: 10.1109/IROS.2017.8202247.

[201] H. Ye, W. Zhu, C. Wang, R. Wu, and Y. Wang. Faster voxelpose: Real-time 3d human pose estimation by orthographic projection. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part VI*, pages 142–159. Springer, 2022.

[202] J. Zhang, W. Li, P. O. Ogunbona, P. Wang, and C. Tang. RGB-D-based action recognition datasets: A survey. *Pattern Recognition*, 60:86–105, 2016. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2016.05.019. URL `https://www.sciencedirect.com/science/article/pii/S0031320316301029`.

[203] L. Zhao and L. Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

[204] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[205] K. Zhou, Y. Dong, K. Wang, W. S. Lee, B. Hooi, H. Xu, and J. Feng. Understanding and resolving performance degradation in deep graph convolutional networks. In

Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pages 2728–2737, 2021.