Learning for perception and control of robots and smart embedded devices

Politecnico di Bari

Department of Electrical and Information Engineering

INDUSTRY 4.0 – INNOVATION

Ph.D. Program

SSD: ING-INF/04-SYSTEMS AND CONTROL ENGINEERING

Final Dissertation

Learning for Perception and Control of Robots and Smart Embedded Devices

---

by

Walter BRESCIA

Supervisors:

Prof. Eng. Luca DE CICCO

Coordinator of Ph.D. Program:

Prof. Eng. Caterina Ciminelli

Course n°37, 01/01/2022-31/12/2024

Politecnico di Bari

LIBERATORIA PER L'ARCHIVIAZIONE DELLA TESI DI DOTTORATO

Al Magnifico Rettore
del Politecnico di Bari

Il sottoscritto **Walter Brescia** nato a **Bari** il **27/02/1996**

residente a **Bari** in via **Siponto n.1** e-mail **walter.brescia@poliba.it**

iscritto al 3° anno di Corso di Dottorato di Ricerca in **Industria 4.0 - Innovazione** ciclo **XXXVII**

ed essendo stato ammesso a sostenere l'esame finale con la prevista discussione della tesi dal titolo:

Learning for Perception and Control of Robots and Smart Embedded Devices

DICHIARA

1) di essere consapevole che, ai sensi del D.P.R. n. 445 del 28.12.2000, le dichiarazioni mendaci, la falsità negli atti e l'uso di atti falsi sono puniti ai sensi del codice penale e delle Leggi speciali in materia, e che nel caso ricorressero dette ipotesi, decade fin dall'inizio e senza necessità di nessuna formalità dai benefici conseguenti al provvedimento emanato sulla base di tali dichiarazioni;
2) di essere iscritto al Corso di Dottorato di ricerca **Industria 4.0 – Innovazione** ciclo **XXXVII**, corso attivato ai sensi del "*Regolamento dei Corsi di Dottorato di ricerca del Politecnico di Bari"*, emanato con D.R. n.286 del 01.07.2013;
3) di essere pienamente a conoscenza delle disposizioni contenute nel predetto Regolamento in merito alla procedura di deposito, pubblicazione e autoarchiviazione della tesi di dottorato nell'Archivio Istituzionale ad accesso aperto alla letteratura scientifica;
4) di essere consapevole che attraverso l'autoarchiviazione delle tesi nell'Archivio Istituzionale ad accesso aperto alla letteratura scientifica del Politecnico di Bari (IRIS-POLIBA), l'Ateneo archivierà e renderà consultabile in rete (nel rispetto della Policy di Ateneo di cui al D.R. 642 del 13.11.2015) il testo completo della tesi di dottorato, fatta salva la possibilità di sottoscrizione di apposite licenze per le relative condizioni di utilizzo (di cui al sito http://www.creativecommons.it/Licenze), e fatte salve, altresì, le eventuali esigenze di "embargo", legate a strette considerazioni sulla tutelabilità e sfruttamento industriale/commerciale dei contenuti della tesi, da rappresentarsi mediante compilazione e sottoscrizione del modulo in calce (Richiesta di embargo);
5) che la tesi da depositare in IRIS-POLIBA, in formato digitale (PDF/A) sarà del tutto identica a quelle consegnate/inviate/da inviarsi ai componenti della commissione per l'esame finale e a qualsiasi altra copia depositata presso gli Uffici del Politecnico di Bari in forma cartacea o digitale, ovvero a quella da discutere in sede di esame finale, a quella da depositare, a cura dell'Ateneo, presso le Biblioteche Nazionali Centrali di Roma e Firenze e presso tutti gli Uffici competenti per legge al momento del deposito stesso, e che di conseguenza va esclusa qualsiasi responsabilità del Politecnico di Bari per quanto riguarda eventuali errori, imprecisioni o omissioni nei contenuti della tesi;
6) che il contenuto e l'organizzazione della tesi è opera originale realizzata dal sottoscritto e non compromette in alcun modo i diritti di terzi, ivi compresi quelli relativi alla sicurezza dei dati personali; che pertanto il Politecnico di Bari ed i suoi funzionari sono in ogni caso esenti da responsabilità di qualsivoglia natura: civile, amministrativa e penale e saranno dal sottoscritto tenuti indenni da qualsiasi richiesta o rivendicazione da parte di terzi;
7) che il contenuto della tesi non infrange in alcun modo il diritto d'Autore né gli obblighi connessi alla salvaguardia di diritti morali od economici di altri autori o di altri aventi diritto, sia per testi, immagini, foto, tabelle, o altre parti di cui la tesi è composta.

Luogo e data **Bari, 21/03/2025** Firma _____

Il sottoscritto, con l'autoarchiviazione della propria tesi di dottorato nell'Archivio Istituzionale ad accesso aperto del Politecnico di Bari (POLIBA-IRIS), pur mantenendo su di essa tutti i diritti d'autore, morali ed economici, ai sensi della normativa vigente (Legge 633/1941 e ss.mm.ii.),

CONCEDE

- al Politecnico di Bari il permesso di trasferire l'opera su qualsiasi supporto e di convertirla in qualsiasi formato al fine di una corretta conservazione nel tempo. Il Politecnico di Bari garantisce che non verrà effettuata alcuna modifica al contenuto e alla struttura dell'opera.
- al Politecnico di Bari la possibilità di riprodurre l'opera in più di una copia per fini di sicurezza, back-up e conservazione.

Luogo e data **Bari, 21/03/2025** Firma _____

Politecnico di Bari

Department of Electrical and Information Engineering

INDUSTRY 4.0 – INNOVATION

Ph.D. Program

SSD: ING-INF/04-SYSTEMS AND CONTROL ENGINEERING

Final Dissertation

Learning for Perception and Control of Robots and Smart Embedded Devices

---

by

Walter BRESCIA

_____

Referees:

Prof. Eng. Carsten Griwodz

Prof. Eng. Antonio Pietrabissa

Supervisors:

Prof. Eng. Luca DE CICCO

_____

Coordinator of Ph.D Program:

Prof. Eng. Caterina Ciminelli

_____

Course n°37, 01/01/2022-31/12/2024

Alla mia famiglia.

Abstract

This Ph.D. thesis studies applications of Deep Learning and Deep Reinforcement Learning in the two main topics of robotics: *perception* and *control*.

Perception is the field of robotics that deals with the robot's ability to acquire, interpret, and understand data from its surrounding environment. This capability is essential for enabling robots to interact effectively with the real world, facilitating tasks such as navigation, object recognition, obstacle avoidance, and context-aware decision-making. Perception relies on a diverse set of sensors that robots use to collect information, including RGB cameras, depth cameras, proximity sensors like LiDAR and millimeter wave (mmWave) radar, inertial sensors for motion and orientation measurements, and environmental sensors that capture data such as temperature, pressure, or sound. These sensors allow robots to create representations of their environment, such as 3D maps or segmented images, which support operations like motion planning, object manipulation, and human-robot collaboration.

In the recent years, the integration of deep learning has revolutionized robotic perception. Traditional methods, based on explicitly programmed algorithms for processing sensory data, have shown limitations in complex or dynamic environments. Deep learning has addressed these limitations through its ability to learn hierarchical representations directly from raw data, generalize to unseen scenarios, and process multi-modal information from diverse sensor types. Advanced models, such as deep neural networks and transformers, have been applied to tasks like object and scene recognition, semantic segmentation, depth estimation, and pose estimation, enabling robots to interpret their surroundings with greater accuracy and efficiency.

The adoption of deep learning in robotic perception has led to a profound integration between advanced hardware and intelligent algorithms. This synergy allows autonomous systems to navigate unfamiliar environments, perform precise manipulations of objects identified through computer vision, and interact with humans by interpreting gestures, voice commands, and behaviours. However, implementing deep learning models in robotics also introduces challenges, such as the need for real-time operation on resource-constrained hardware and the reliance on high-quality datasets for

model training.

The first part of this thesis proposes an approach, based on Conditional Generative Adversarial Network (cGAN), to take advantage of the robustness of mmWave sensors to environmental condition, occlusions, and obstacles and produce readable information in the form of depth images. In particular, the proposed techniques maps sparse and noisy mmWave point clouds into depth images to leverage the usability of such data format and all the related techniques. The approach is validated on a dataset collected using a mobile robot equipped with mmWave sensor and a depth camera which serves as ground truth for the training.

A detailed study is then carried out on mmWave point clouds. Using a motion capture system for precise pose estimation, a dataset comprising 12 million points is gathered in indoor scenarios. Each point in the dataset is classified into one of two categories: *true points* or *noise points*. The use of the motion capture system ensures sub-millimetric accuracy for the labelling process. The dataset also provides the distance of the closest obstacle to each point, enabling the use of regression techniques for denosing purposes.

Then, following this work, a benchmark of the state of the art techniques for point cloud elaboration is conducted on the task of mmWave point cloud denoising. The poor performance on the task, stemming from the peculiarities of such data, highlight a gap in the state of the art of point cloud processing. On this basis, a graph-based transformer architecture is proposed to elaborate such point clouds and effectively identify noise points and true points. The proposed technique analyses point clouds both from a temporal and geometrical point of view, highlighting how the sparsity of such data, which impairs the state of the art models, is the key resource in the proposed technique to effectively denoise the data.

The last chapter of this part deals with Unmanned Aerial Vehicles (UAVs). Commonly denoted as drones, this robots are being increasingly adopted in many applications such as surveillance, disaster response, environmental monitoring, live drone broadcasting, etc. This chapter introduces APEIRON, a rich multimodal aerial dataset collecting *perception data* from a stereocamera and an event based camera sensor, along with measurements of*wireless network links* obtained using an LTE module. This dataset effectively bridges many robotics fields, from perception, to telecommunication and control, providing network data, such as bandwidth maps, raw sensors data linked to GPS coordinates and low level data from the drone, creating the basis for applications at the intersection of such fields.

Control is a fundamental field of robotics, encompassing the techniques and strategies that enable robots to perform tasks by regulating their movements and interactions with the environment. At its core, control ensures that a robot follows desired trajectories and executes tasks accurately,

whether it is navigating through a complex environment, manipulating objects, or coordinating with other robots. Traditional control methods, such as PID controllers and model-based approaches, rely heavily on accurate mathematical models of the robot's dynamics and the environment. While effective in many scenarios, these methods can struggle with the complexity, unpredictability, and variability of real-world applications.

Recent advances in machine learning, particularly in Reinforcement Learning (RL), have revolutionized the field of robotic control. Deep reinforcement learning (DRL) integrates the strengths of deep neural networks and RL, enabling robots to learn optimal control policies directly from raw sensor data or high-dimensional state spaces. By interacting with the environment, a robot trained with DRL can iteratively improve its performance through trial and error, discovering control strategies that maximize a given reward signal. This has proven particularly effective in tasks like dynamic manipulation, complex locomotion, and multi-agent coordination, where designing explicit control strategies is challenging.

However, the application of DRL in robotics is not without challenges. Safety is a critical aspect, especially in real-world scenarios where failures can lead to equipment damage, safety risks, or undesirable outcomes. This has led to the emergence of Safe Reinforcement Learning (SRL), a subfield of RL focused on ensuring safety during the learning and deployment phases. SRL introduces constraints and risk-aware mechanisms into the learning process, allowing robots to explore and optimize their behaviour while minimizing the likelihood of catastrophic failures. Techniques in SRL include the use of constrained optimization, risk-sensitive reward functions, and safe exploration strategies, all of which are essential to deploy DRL in safety-critical robotic applications.

The second part of this thesis deals with Reinforcement Learning and Safe Reinforcement Learning techniques for controlling industrial robots.

The first chapter of this part examines state-of-the-art Reinforcement Learning algorithms for pose regulation of a wheeled industrial platform, specifically a *four-wheel steering, four-wheel driving* robot. This robot's multiple actuators provide robustness to faults, making it ideal for industrial applications, but its complex dynamics and kinematics pose significant control challenges, even for learning-based approaches. A benchmark of DRL methods reveals that effective control is hindered by the robot's need for precise wheel coordination. Untrained neural networks struggle to achieve this, leading to poor learning outcomes. To address this, two techniques are introduced: *Episodic Noise*, which helps useful action subsets emerge early in training, and the *Difficulty Manager*, which adjusts goals to match the agent's current capabilities. These tools enable the successful training of a control policy within a few hundred epochs. An ablation study highlights the critical role of effective exploration strategies and curriculum learning in developing controllers for such complex

systems.

The final chapter focuses on the control of a Drivable Vertical Mast Lift (DVML) to enable autonomous navigation while maintaining essential safety constraints. DVMLs are industrial vehicles widely used in applications such as logistics and smart agriculture, allowing operators in an elevated basket to access hard-to-reach work sites. However, improper use of these vehicles easily exposes operators to potential accidents, and therefore they are associated with safety regulations and laws. This chapter explores advancements in Safe RL from a practical perspective, applying several state-of-the-art algorithms to endow a DVML with autonomous driving capabilities. The study highlights that, while benchmark environments effectively validate Safe RL methodologies as proof-of-concept, they often fail to bridge the gap between these environments and real-world applications. This strongly limits a broader adoption of Safe RL methods in industrial use cases, highlighting the need for practical advancements to align these techniques with real-world requirements.

Abstract

(Italian)

Questa tesi di dottorato studia le applicazioni del Deep Learning e del Deep Reinforcement Learning nei due principali ambiti della robotica: *percezione* e *controllo*.

La percezione è il campo della robotica che si occupa della capacità del robot di acquisire, interpretare e comprendere le informazioni provenienti dall'ambiente circostante. Questa abilità è essenziale per consentire ai robot di interagire efficacemente con il mondo reale, facilitando attività come la navigazione, il riconoscimento degli oggetti e l'evitamento degli ostacoli. La percezione si avvale di un insieme diversificato di sensori utilizzati per raccogliere informazioni, tra cui telecamere RGB, telecamere di profondità, sensori di prossimità come LiDAR e radar ad onde millimetriche (mmWave), sensori inerziali per misurazioni di movimento e orientamento, e sensori ambientali che rilevano dati come temperatura e pressione. Questi sensori permettono ai robot di creare rappresentazioni dell'ambiente circostante, come mappe 3D o immagini segmentate, che permettono in seguito operazioni come la pianificazione del movimento, la manipolazione di oggetti e la collaborazione tra uomo e robot.

Negli ultimi anni, l'introduzione del deep learning ha rivoluzionato il campo della percezione nella robotica. I metodi tradizionali, basati su algoritmi programmati esplicitamente per elaborare dati sensoriali, sono spesso associati a forti limiti, soprattutto in ambienti complessi o dinamici. Il Deep Learning ha dimostrato di poter superare questi limiti grazie alla sua capacità di apprendere rappresentazioni utili direttamente dai dati grezzi, generalizzare a scenari mai esplorati in precedenza e processare informazioni multi-modali provenienti da diversi tipi di sensori. Modelli avanzati, come reti neurali profonde e transformers, sono stati testati in task quali il riconoscimento di oggetti e scene, la segmentazione semantica, la stima della profondità e del posizionamento, consentendo ai robot di interpretare l'ambiente circostante con maggiore accuratezza ed efficienza.

L'adozione del deep learning nella percezione robotica ha portato a una profonda integrazione tra hardware avanzato e algoritmi intelligenti. Questa sinergia permette ai sistemi autonomi di navigare in ambienti sconosciuti, eseguire manipolazioni precise di oggetti identificati attraverso la visione

artificiale e interagire con gli esseri umani interpretando gesti, comandi vocali e comportamenti. Tuttavia, l'implementazione di modelli di deep learning nella robotica introduce anche sfide, come la necessità di operazioni in tempo reale su hardware con risorse limitate e la dipendenza da dataset di alta qualità per l'addestramento dei modelli.

La prima parte di questa tesi propone un approccio basato su cGAN per sfruttare la robustezza dei sensori ad onde millimetriche rispetto a condizioni ambientali, occlusioni e ostacoli, e generare informazioni maggiormente fruibili sotto forma di immagini di profondità. In particolare, la tecnica proposta mappa le nuvole punti sparse e rumorose di sensori mmWave in immagini di profondità per sfruttare la praticità di questo formato dati e tutte le tecniche ad esso correlate. L'approccio è validato su un dataset raccolto utilizzando un robot mobile equipaggiato con sensori mmWave e una telecamera di profondità, che viene utilizzata come ground truth in fase di addestramento.

Il capitolo successivo effettua un'analisi dettagliata di questo tipo di nuvole punti. Utilizzando un sistema di motion capture per ottenere una stima precisa delle posizioni, viene raccolto un dataset di 12 milioni di punti in scenari indoor. Ogni punto nel dataset viene classificato in due categorie: *punti reali* e *rumorosi*. L'uso del sistema di motion capture assicura un'accuratezza sub-millimetrica nel processo di labelling. Il dataset fornisce, inoltre, la distanza dell'ostacolo più vicino per ciascun punto, permettendo l'uso di tecniche di regressione per il task di denoising.

Estendendo questo lavoro, viene condotto un benchmark delle tecniche note allo stato dell'arte per l'elaborazione di nuvole di punti sul task di denoising delle nuvole di punti di sensori mmWave. I risultati del benchmark evidenziano la difficoltà delle tecniche dello stato dell'arte nell'elaborazione di questo tipo di nuvole di punti.

Sulla base di questi risultati, viene proposta un'architettura transformer basata sull'elaborazione di grafi per processare tali nuvole punti e identificare efficacemente punti di rumore e punti reali. La tecnica proposta analizza le nuvole punti sia da un punto di vista temporale che geometrico, dimostrando come la sporadicità di tali dati, che rende i modelli esistenti inefficaci, diventi una risorsa chiave nella tecnica proposta per il denoising di questo tipo di informazioni.

L'ultimo capitolo della prima parte riguarda i veicoli aerei senza pilota (UAV), chiamati anche droni, sempre più spesso adottati in numerose applicazioni quali sorveglianza, monitoraggio ambientale, trasmissione live, ecc. Questo capitolo introduce APEIRON, un dataset aereo multimodale che raccoglie dati di percezione da una stereocamera e da un sensore a telecamera event-based, insieme a misurazioni dei collegamenti di rete wireless ottenute utilizzando un modulo LTE. Questo dataset collega efficacemente molti campi della robotica, dalla percezione, alla telecomunicazione e al controllo, fornendo dati di rete, come mappe di larghezza di banda, dati grezzi di numerosi sensori e coordinate GPS e dati di basso livello del drone, creando le basi per applicazioni all'intersezione di

questi campi.

Insieme alla percezione, il controllo è un campo fondamentale della robotica, che comprende le tecniche e le strategie che consentono ai robot di svolgere compiti compiend movimenti e interagendo con l'ambiente. Le tecniche di controllo garantiscono che un robot segua traiettorie desiderate ed esegua i compiti con precisione, sia che si tratti di navigare in ambienti complessi, manipolare oggetti o coordinarsi con altri robot. I metodi di controllo tradizionali si basano fortemente su modelli matematici accurati delle dinamiche del robot e dell'ambiente. Sebbene efficaci in molte situazioni, questi metodi possono incontrare difficoltà di fronte alla complessità, all'imprevedibilità e alla variabilità delle applicazioni reali, specialmente quando derivare un modello matematico diventa complesso o il modello matematico non è sufficientemente accurato.

Recenti progressi nell'ambito dell'apprendimento automatico, in particolare nel Deep Reinforcement Learning, hanno rivoluzionato il campo del controllo robotico. Il Deep Reinforcement Learning combina le potenzialità delle reti neurali profonde e del Reinforcement Learning, consentendo ai robot di apprendere policy di controllo ottimali, mappando direttamente i dati sensoriali grezzi in azioni di controllo. Interagendo con l'ambiente, gli algoritmi di DRL utilizzano un processo iterativo di *trial and error* per migliorare le proprie prestazioni attraverso, individuando strategie di controllo che massimizzano una data *reward function*. Queste tecniche si sono dimostrate particolarmente efficaci in numerosi compiti, tra cui la manipolazione, la locomozione in ambienti complessi e la coordinazione multi-agente, dove la progettazione di strategie di controllo esplicite risulta un processo particolarmente complicato ed incline ad errori.

Tuttavia, l'applicazione del DRL nel campo della robotica non è priva di sfide. La sicurezza rappresenta un aspetto cruciale, specialmente in scenari reali dove i fallimenti possono portare a danni alle apparecchiature, rischi per la sicurezza o risultati indesiderati. Questo ha portato alla nascita del cosidetto Safe Reinforcement Learning (SRL), una branca del RL focalizzata sulla sicurezza durante le fasi di apprendimento e implementazione. Il SRL introduce vincoli e meccanismi orientati alla gestione del rischio nel processo di apprendimento, consentendo ai robot di esplorare e ottimizzare il loro comportamento minimizzando la probabilità di fallimenti o danneggiamenti. Le tecniche nel SRL includono l'ottimizzazione vincolata, funzioni di ricompensa che includono una componente legata al rischio di violazione di vincoli e strategie di esplorazione sicura, tutte essenziali per l'implementazione del DRL in applicazioni robotiche critiche per la sicurezza.

La seconda parte di questa tesi riguarda le tecniche di Reinforcement Learning e Safe Reinforcement Learning per il controllo di robot industriali.

Il primo capitolo di questa parte esamina gli algoritmi di Reinforcement Learning allo stato dell'arte per il controllo della posizione di una piattaforma industriale mobile, in particolare di

un robot a quattro ruote sterzanti e motrici. I numerosi attuatori di questo robot forniscono un'elevata robustezza ai guasti, rendendolo ideale per applicazioni industriali, ma la sua dinamica e la sua cinematica particolarmente complesse rappresentano una sfida significativa, sia per approcci tradizionali sia per approcci basati sull'apprendimento automatico. Un benchmark delle metodologie di DRL mostra come anche queste tecniche faticano nel controllo di questo robot. Infatti, la coordinazione di tutti gli attuatori è necessaria per un controllo efficace di questo robot. Tuttavia, tale coordinazione rappresenta una sfida complessa per le reti neurali all'inizio del processo di addestramento, portando a scarsi risultati. Per affrontare questo problema, vengono proposte due tecniche: l'*Episodic Noise*, che aiuta ad esplorare efficacemente lo spazio delle azioni e a far emergere un buon controllore già nelle prime fasi di addestramento, e il *Difficulty Manager*, che regola la difficoltà degli obiettivi in base alle capacità correnti dell'agente. Questi strumenti consentono l'addestramento di una policy di controllo che risulta efficace nel controllo del robot in poche centinaia di epoche. Lo studio di ablazione condotto evidenzia il ruolo cruciale di strategie di esplorazione efficaci e di tecniche di curriculum learning nello sviluppo di controllori basati sull'apprendimento autonomo per sistemi così complessi.

L'ultimo capitolo si concentra sulla navigazione autonoma di un Drivable Vertical Mast Lift (DVML) garantendo il rispetto dei vincoli di sicurezza. I DVML sono veicoli industriali ampiamente utilizzati in applicazioni come la logistica e l'agricoltura intelligente, consentendo agli operatori situati in un cestello elevato di accedere a siti di lavoro difficili da raggiungere. Tuttavia, un utilizzo improprio di questi mezzi espone facilmente gli operatori a potenziali incidenti e sono per questo associati a norme e leggi di sicurezza. Questo capitolo esplora i progressi nel Safe RL da un punto di vista pratico, applicando diversi algoritmi nello stato dell'arte per dotare un DVML di capacità di guida autonoma. Lo studio evidenzia come, utilizzando ambienti *proof-of-concept* per la validazione delle metodologie di Safe RL, si va a costituire un divario tra le applicazioni reali e le stesse metodologie proposte. Questo limita fortemente una più ampia adozione delle metodologie di Safe RL in scenari industriali, evidenziando la necessità di validare queste metodologie in ambienti più vicini agli utilizzi pratici per allineare le tecniche ai requisiti del mondo reale.

Table of Contents

List of Figures . . . . . **iv**

List of Abbreviations . . . . . **vi**

Scientific Contributions . . . . . **vii**

1 Introduction **1**

1.1 Robotics 2

1.2 Artificial Intelligence and Machine Learning 3

1.3 Edge-AI 6

1.4 Thesis Outline 7

2 Background **9**

2.1 Robotics 9

2.1.1 Perception 11

2.1.2 Control 13

2.2 Machine Learning 16

2.2.1 Artificial Neural Networks 18

2.2.2 Reinforcement Learning 21

2.2.3 Safe Reinforcement Learning 25

2.3 Smart Embedded Devices 26

2.3.1 Post-Training Quantization 27

2.3.2 Quantization-Aware Training 27

I Perception **29**

3 Point2Depth: a GAN-based Contrastive Learning Approach for mmWave Point Clouds to Depth Images Transformation **30**

3.1 Introduction . . . 30
3.2 Related work . . . 31
3.2.1 Contrastive Learning . . . 31
3.2.2 Conditional Generative Adversarial Networks . . . 32
3.2.3 Point Cloud elaboration . . . 32
3.2.4 RGB to Depth . . . 32
3.3 mmWave Point Clouds to Depth Images . . . 33
3.3.1 RGB to Depth Image cGAN . . . 35
3.3.2 Point2Latent . . . 36
3.4 Results . . . 37
3.4.1 Dataset . . . 39
3.4.2 Performance Analysis . . . 39
3.4.3 Experiments . . . 43
3.5 Quantizing Point2Depth . . . 46
3.6 Concluding Remarks . . . 51

4 MilliNoise: a Millimeter-wave Radar Sparse Point Cloud Dataset in Indoor Scenarios **53**

4.1 Introduction . . . 53
4.2 Background on mmWave radars . . . 54
4.3 Related Work . . . 56
4.4 The MilliNoise dataset . . . 58
4.4.1 Capturing MilliNoise . . . 58
4.4.2 Point Cloud Pre-Processing . . . 60
4.4.3 Data Labeling . . . 61
4.4.4 Dataset features . . . 62
4.4.5 Dataset Organization . . . 62
4.4.6 Dataset Tools . . . 63
4.5 Concluding Remarks . . . 64

5 GT-MilliNoise: Graph Transformer for Point-wise Denoising of Indoor Millimeter-Wave Point Clouds **65**

5.1 Introduction . . . 65
5.2 Background . . . 67
5.2.1 FMCW Radar and mmWave Point Clouds (PCs) . . . 67
5.2.2 Point Clouds Processing . . . 68
5.2.3 Related Work on mmWave Point Clouds Denoising . . . 69
5.3 mmWave Point Cloud Denoising . . . 70
5.3.1 Proposed GT Approach . . . 71
5.3.2 GT Architecture . . . 73
5.4 Implementation . . . 76
5.4.1 Training . . . 76
5.4.2 Evaluation Metrics . . . 77
5.4.3 Bechmarking Methods . . . 78
5.4.4 GT Architecture Details . . . 79
5.5 Results . . . 79
5.5.1 Understating of Accuracy and Geometric Metrics . . . 79
5.5.2 Overall Results . . . 82
5.5.3 Generalization to unseen scenarios - K-Folds Results . . . 83
5.5.4 3D Space Exploration Discussion . . . 83
5.5.5 Ablation Studies . . . 87
5.5.6 Computational Complexity . . . 92
5.5.7 Limitations and Future Work . . . 92
5.5.8 Concluding Remarks . . . 92

6 APEIRON: a Multimodal Drone Dataset Bridging Perception and Network Data in Outdoor Environments **94**

6.1 Introduction . . . 94
6.2 Related Work . . . 95
6.2.1 Network-related datasets . . . 96
6.2.2 Perception datasets . . . 96
6.3 Methodology . . . 97
6.3.1 Data collection platform . . . 98
6.3.1.1 Software stack . . . 100
6.4 The APEIRON dataset . . . 102

6.4.1 Scenarios . . . 102
6.4.2 Dataset format . . . 103
6.4.3 Analysis of the dataset . . . 104
6.4.4 Perception . . . 104
6.5 Concluding Remarks . . . 106

II Control **108**

7 Sample Efficient Reinforcement Learning for pose regulation of mobile robot **1**
7.1 Introduction . . . 1
7.2 Background . . . 2
7.2.1 Classical Control of a 4WS4WD . . . 2
7.2.2 RL-based Control of a 4WS4WD . . . 3
7.3 Problem Statement . . . 4
7.4 Methodology . . . 7
7.4.1 *Difficulty Manager (DM)* . . . 7
7.4.2 *Episodic Noise* . . . 9
7.5 Results . . . 9
7.5.1 Overall Performance . . . 12
7.5.2 Linear and Angular Distances Coverage . . . 14
7.5.3 Maneuverability Exploitation . . . 16
7.5.4 Behaviour's Analysis . . . 16
7.5.5 Discussions . . . 16
7.6 Concluding Remarks . . . 18

8 Safe Reinforcement Learning for Autonomous Navigation of a DVML **19**
8.1 Introduction . . . 19
8.2 Related Work . . . 21
8.2.1 RL Methodologies for safety critical applications . . . 22
8.2.2 RL for Industrial applications . . . 23
8.3 State of the art . . . 24
8.4 Problem statement . . . 26
8.4.1 Autonomous Navigation of a DVML . . . 27
8.5 Environment Setting . . . 29
8.6 Results . . . 29

8.6.1 Overall results 30
8.6.2 Detailed analysis 32
8.6.3 Discussion 34
8.7 Concluding Remarks 35

9 Conclusions and Future Research Directions 36

List of Figures

1.0.1 Archytas' pigeon 1

2.1.1 Types of encoders 11
2.1.2 Control Loop 14
2.2.1 Interaction between agent and environment in a Markov Decision Process 22

3.3.1 Point2Depth Architecture 34
3.3.2 RGB2Depth training scheme 34
3.3.3 Point2Depth Overall Training Scheme 34
3.3.4 Point2Latent training process 37
3.3.5 Point2Latent Structure 38
3.4.1 Training Losses for Generator (a), Discriminator (b) and Point2Latent (c) 41
3.4.2 Real Colour Image (a), Real Depth Image (b), SA-A prediction (c) and RN-N prediction (d) 42
3.4.3 Test 1 44
3.4.4 Test 1 - Model predictions 45
3.4.5 Test 2 47
3.4.6 Test 2 - Model predictions 48
3.5.1 Point2Depth vs QPoint2Depth - frame 3 51
3.5.2 Point2Depth vs QPoint2Depth - frame 367 52

4.2.1 Sensor Pipeline employed to produce PCs from sensor's readings 55
4.4.1 The MilliNoise dataset acquisition system 58
4.4.2 Scenario definition 59
4.4.3 Example runs of each of the six scenes available in the MilliNoise dataset. Obstacles are shown in red. Clean (noise) points are shown in yellow (blue). The trajectory followed by the robot is shown with a dashed line. 60

4.4.4 The directory structure of the dataset. 63

5.3.1 Proposed pipeline architecture for PC denoising composed of two main blocks: *Temporal* and *Geometric* and example how on the PC is converted to graph in each block 71
5.3.2 Example of how the edges are built in the temporal graph $\mathcal{G}_{\mathcal{T}}$ processed by the *Temporal* block. The point in blue (yellow) is an example of a consistent (sporadic) point in time. 73
5.3.3 Schematic of a *Geometric* block layer. 74
5.5.1 Denoising results of Scene 1 (*Fold-123* run 4). 80
5.5.2 Denoising results of Scene 4 (*Fold-4* run 70). 81
5.5.3 Features PCA of each models (PCs with $W = 12$ and plotted from the robot perspective). 84
5.5.4 Prediction of each model (PCs with $W = 12$, in the robot RF). 85
5.5.5 Zoom-in visualization in a region of interest from Figure 5.5.3 86
5.5.6 Comparison of pseudo-temporal consistency (darker colour represents a stronger temporal consistency) with the features learned on *Temporal* block projected to 3 and 1 dimensions. Figure from two PCs examples with $\mathcal{W} = 12$ acquired from the robot reference system. 89

6.3.1 Hexarotor drone and sensors placement 97
6.3.2 Software and connection architecture composed by: the Jetson Xavier on-board of the UAV, the remote server used as target for the uplink/downlink network traces, the user SSH interface used to start/stop and monitor the collection process via a remote shell. 100
6.4.1 Example scenarios and trajectories 102
6.4.2 Network traces: continuous (dashed) lines refer to uplink (downlink) measurements. 105
6.4.3 Perception data (run IND-4) 105
6.4.4 Localization using GPS or ZED Positional Tracking 107

7.3.1 The considered robot 5
7.4.1 Difficulty Manager (DM) 8
7.5.1 RL Algorithms Comparison 10
7.5.2 Overall Performance 13
7.5.3 CCDFs on linear and angular distance 15
7.5.4 Representative successful trajectories 17

8.4.1 28
8.6.1 Overall Training Performance 31
8.6.2 Average velocity when constraint violations happen, during Tests 32
8.6.3 CCDF of Minimum Distance from Obstacles during Tests 33
8.6.4 Vibration evolution of CPO agent 33

List of Abbreviations

AdamBA Adaptive Momentum Boundary Approximation Algorithm

AI Artificial Intelligence

AMLWP Aerial Mobile Lifting Working Platform

APPO Augmented Proximal Policy Optimization

cGAN Conditional Generative Adversarial Network

CMDP Constrained Markov Decision Process

CPO Constrained Policy Optimization

CPS Cyber-Physical Systems

CRPO Constrained Rectified Policy Optimization

CUP Constrained Update Projection

CVPO Constrained Variational Policy Optimization

DD Differential Drive

DDPG Deep Deterministic Policy Gradient

DGCNN Dynamic Graph Convolution

DM Difficulty Manager

DRL Deep Reinforcement Learning

DVML Driveable Vertical Mast Lift

FMCW Frequency-Modulated Continuous Waves

FOCOPS First Order Constrained Optimization in Policy Space

FPS farthest-point-sampling

GAN Generative Adversarial Network

GNN Graph Neural Network

ISSA Implicit Safety Set Algorithm

LiDAR Light Detection and Ranging

MDP Markov Decision Process

mmWave millimeter wave

MPC Model Predictive Control

NN Neural Network

P3O Penalized Proximal Policy Optimization

PC Point Cloud

PPO Proximal Policy Optimization

ReLU Rectified Linear Unit

RL Reinforcement Learning

RL Reinforcement Learning

SAC Soft Actor Critic

Safe RL Safe Reinforcement Learning

SAUTÉ Safety AUgmenTEd

SG SafetyGym

SIMMER Safe policy IMproveMEnt for RL

TRPO Trust Region Policy Optimization

Scientific Contributions

The scientific contributions that summarize all the activities carried out during the Ph.D. are listed below.

International Conferences:

- W. Brescia, L. De Cicco and S. Mascolo, "Sample-Efficient Reinforcement Learning for Pose Regulation of a Mobile Robot," 2022 11th International Conference on Control, Automation and Information Sciences (ICCAIS), Hanoi, Vietnam, 2022, pp. 42-47.
- W. Brescia, A. Maci, S. Mascolo and L. De Cicco, "Safe reinforcement learning for autonomous navigation of a driveable vertical mast lift", 2023 22nd World Congress of the International Federation of Automatic Control (IFAC WC), Yokohama, Japan, 56(2), pp.9068-9073.
- W. Brescia, G. Roberto, V. A. Racanelli, S. Mascolo and L. D. Cicco, "Point2Depth: a GAN-based Contrastive Learning Approach for mmWave Point Clouds to Depth Images Transformation," 2023 31st Mediterranean Conference on Control and Automation (MED), Limassol, Cyprus, 2023, pp. 529-534.
- W. Brescia, P. Gomes, L. Toni, S. Mascolo, and L. De Cicco. 2024. MilliNoise: a Millimeter-wave Radar Sparse Point Cloud Dataset in Indoor Scenarios. In Proceedings of the 15th ACM Multimedia Systems Conference (MMSys '24). Best ODS Paper Awarded, 422–428.
- N. Barone, W. Brescia, S. Mascolo, and L. De Cicco. 2024. APEIRON: a Multimodal Drone Dataset Bridging Perception and Network Data in Outdoor Environments. In Proceedings of the 15th ACM Multimedia Systems Conference (MMSys '24), 401–407.

International Journals:

- P. Gomes, W.Brescia, S. Mascolo, L. Toni, L. De Cicco, "GT-MilliNoise: Graph Transformer for Point-wise Denoising of Indoor Millimeter-Wave Point Clouds" (submitted to ACM Transactions on Multimedia Computing, Communications, and Applications)

Chapter 1

Introduction

Creating intelligent artificial companions that could help humans with repetitive and dangerous tasks has always driven the curiosity of many scholars and academics.

The first attempts at creating autonomous robots date back to Archytas, an ancient Greek mathematician from the ancient city of Taras (the modern city of Taranto). The story goes that Archytas built what can be called the first flying robot: a wooden steam-propelled bird (Figure 1.0.1). Despite the lack of proofs for this particular invention, the concept of robots and automata was already widespread in the ancient Greece. The first well described automata is *Herone's theatre*: a contraption that, thanks to counterweights, sand and ropes, brings Dionysus to life in a theatrical representation. During the years, several texts described contraptions and machinery from all over the world, employed in a wide range of applications, from entertainment to war purposes.

Today, automation is the term that refers to a set of tools and techniques employed to make a process act out of its own "will". It is often associated to the field of robotics, as it enables such machines to carry out tasks without human supervision. In fact, automation and robotics are strongly interleaved, as robotics provides the physical tools that enact the automatic process.

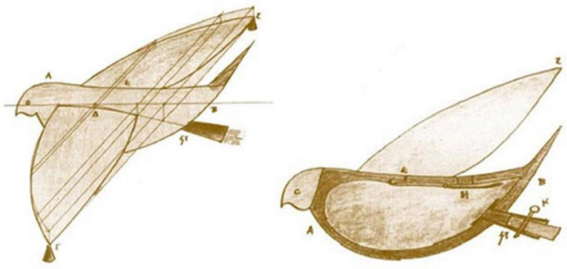


Figure 1.0.1: Archytas' pigeon

1.1 Robotics

Robotics can be described as the study and design of three main components: the hardware part, which physically builds the robot; the electronics of the robot, which powers the components; and the software, which implements the logic and the rules under which the robot should operate and that control the robot itself.

Designing the logic and the set of rules for a robotic machine that should operate in a given environment is a difficult process, which requires studying and testing all the phases of the process that the robot should follow. It requires studying how the robot moves (*kinematics* and *dynamics*), the environment in which it operates and the possible objects with which it will interact.

However, designing such logic is not enough: the design process requires finding out possible issues and improving iteratively until the final result satisfies the requirements.

Depending on the contexts, this process can last few days to months and years and the iterative process can involve simple fixing, redesigning, and, sometimes, even repairing possibly damaged hardware and electronic components.

The main research field that tackles the design of techniques and tools to implement the logic is the *control theory*. Control theory is a research field that sets its roots in the control engineering and applied mathematics fields. Its main objective is to develop tools and algorithms to control dynamical systems.

Typically, these algorithms observe the state of the system and compute a form of error with respect to the desired trajectory. Then, from the error, the *controller* will compute a *control input*. Designing these types of controllers requires modelling the dynamical system in a way that, given the control input, it is possible to compute the state at the next step. The result of the modelling process is, then, an analytical tool which expresses the evolution of the dynamical system in time.

However, modelling a dynamical system can be a difficult and time consuming task, depending on the peculiarities of the system itself.

In fact, this process requires studying the dynamics of the systems and finding a mathematical model that describes how the system is expected to react when stimulated by any control action or external disturbances, which might not always be possible. The most common approach to derive the model is to simplify the dynamics to obtain or identify a linear model (i.e. a model with linear differential equations) that binds a control input to the output of the system. In order to control the machines associated to linear models it is possible to leverage the whole research field of linear control theory. However, as systems' complexity increases, the models tend to be much more complicated and non-linear. Note that almost all real world systems are non-linear, which means that performance of linear controllers will be reduced on most real problems. Being governed

by non-linear differential equations, this class of problems requires a careful and sensitive design process of the controller. This often produces a controller which is strongly focused on the particular system and is hardly adaptable to variations of the same system.

As systems grow in complexity, it often becomes necessary to leverage more sensors to capture the surroundings, f.i. to identify obstacles. However, integrating sensors with the system makes the modelling process increasingly sensitive and prone to errors, due to the necessity of leveraging the information the sensors provide. Moreover, introducing sensors requires translating the raw data coming from sensors into a form of error which can be read by the controller, which extends the design process.

In fact, expressing the data coming from sensors such as LiDARs or cameras in the form of an error with respect to a given task requires knowledge in the domain of the sensor itself, f.i. image processing and computer vision. This requirement makes the design of a controller a multi-disciplinary task.

In the context of mobile robots, i.e. all the robots that are capable of moving in an environment without being fixed to a given position, the multidisciplinarity of this task is exacerbated by the necessity of the robot of being able to know its position and the position of possible (static and dynamic) obstacles in their way.

From these difficulties a new branch of control theory, named *intelligent control*, emerges to try to leverage new techniques in the context of controls.

In particular, intelligent controls take advantage of some of the tools from *Artificial Intelligence* (AI) and *Machine Learning* (ML). For instance, during the modelling phase, one could use *Neural Networks* to try to estimate the dynamics of the system. Other approaches employ neural networks to compute the control input directly from the observation (i.e. raw data coming from sensors), leveraging their capabilities at processing any kind of information.

However, these types of tools do not provide any guarantee on the performance and safety of the system, which makes the whole research field particularly lively.

Despite their employment in control theory, artificial intelligence and machine learning techniques have a different background.

1.2 Artificial Intelligence and Machine Learning

Artificial Intelligence sets its roots in many different scientific fields and tackles many different aspects. It is a science that studies human intelligence and attempts to mimic it in machines. In antiquity, philosophers tried to study the way the mind formulates logical reasoning by using formal

rules. The study of formal rules and how these can be mapped to the processes of the human mind led to Alan Turing's studies, in which he formulates how machines could employ such rules just by rearranging "0" and "1" symbols.

His studies strongly influenced scholars of that period. In fact, in 1943, Warren McCulloch and Walter Pitts published "A Logical Calculus of the ideas Immanent in Nervous Activity" [1], in which they describe how neural activity can be approximated with propositional logic, by proposing the "McCulloch-Pitts neuron", the first mathematical model of a neural network.

Later, in 1957, Herbert A. Simon, J. C. Shaw, and Allen Newell first introduced the so-called "General Problem Solver" (GPS), later published in [2], a program imitating human intelligence to solve a subclass of logical puzzles. A major step toward AI as we know it is made by two works in 1962: the Frank Rosenblatt's *perceptron* [3] and Block's proof of the *Perceptron Convergence Theorem* [4]. In particular, the latter formalized how the learning algorithm is capable of adjusting weights between connections in the network to match any information observed in input, given such a match existed in the first place. Later on, in 1969, Minsky and Papert formalized in "Perceptrons" [5] how these simple forms of neural networks could actually learn anything these networks are capable of representing, arguing, though, that these networks' representation capabilities are very small. In the very same year, Bryson and Ho, in their book "Applied Optimal Control" [6], introduced the *back-propagation*, an algorithm to learn patterns and rules from data leveraging a multi-layer perceptron, hence providing the basis for modern machine learning.

In an attempt to adopt the scientific method and bring ML and AI methodologies in a rigorous form, many studies in the second half of the 1960s, specifically in the context of speech recognition, have started leveraging *Hidden Markov Models* (HMMs). An HMM is a tool that enables modelling an observable process Y whose output is only dependent, in a known way, on a non-observable process X. An important aspect of adopting the HMMs methodology is related to a requirement of such tool, that is, the output of Y at time $t = t_0$ must be influenced only by the outcome of X at time $t = t_0$. Furthermore, all outputs of X and Y at time $t < t_0$ must be independent of the output of Y at time $t = t_0$, given X at time $t = t_0$. Adopting HMMs has enabled researchers to leverage such a framework and support the engineering claims made in their works.

Today, AI has grown with these tools, developing much more complex techniques. Neural networks now adopt non-linearities to improve the expressiveness of the architecture, that is, its capability of representing complex relations between the input and the expected output.

At the basis of modern neural network structures there is the *artificial neuron*, which represents the mathematical model of what scientists observe in human (and, more in general, in mammals). In *artificial neural networks* (ANNs), neurons are organized in layers and connected with each other in

a hierarchical structure. Each relation between neurons is then weighted by a parameter (or *weight*), which allows to strengthen or weaken the relation, just like in the biological twin. In order to get the output from this structure, the information is provided in the input layer and then, iteratively up to the last layer, the information is processed by the weights and passed to the next layer. Finally, the last layer represents the output of the network.

These algorithms now cover a great number of topics, such as speech recognition, computer vision and image processing, game playing, robotics, search engines, recommender systems and so on.

Overall, we can summarize the ML algorithms in three main categories: *supervised*, *unsupervised* and *reinforcement* learning.

Supervised Learning refers to all the algorithms that have to predict, given some data in input, the exact value associated to said data. This value is called *label* and represents the value that the ML algorithm is expected to predict. The distance between the label and the predicted value represents the error (or *loss*) to be minimised.

Unsupervised Learning is the set of algorithms that analyses data in search of patterns and properties. In this case, no label is provided and an auxiliary metric (f.i. a similarity metric) is leveraged to estimate the performance of the tool and drive its improvements.

Reinforcement Learning algorithms are usually involved in a process that has to be controlled. In this type of algorithms, the output represents the control action, while, most of the times, the data observed are produced by sensors used to gather information from the process that has to be controlled. In this case, the loss describes the performance with respect to the specific task and it is up to the engineer to design an effective loss that, when minimised, guarantees that the task is carried out as expected.

In each setting, ML techniques have showed to reach incredible performance leveraging many different forms of data such as images, text, video or voice. Thanks to their efficacy, many forms of AI are now a key part of many commercial products, providing services from everyday utilities to autonomous driving cars.

Most popular models associated with incredible performance are also associated with incredibly prohibitive hardware requirements, i.e. high-performing CPUs, GPUs and memory. This makes these models inaccessible to most users. This led to the development of cloud-based platforms that provide these models in the form of services, reducing the requirements to simple access to the Internet.

However, not all AI applications can come to terms with such a requirement while other applications, usually the ones that have to deal with safety concerns and real-time requirements,

cannot put up with any delay or potential connection issues that might be due to the employed communication network. These conditions are exacerbated in *edge devices*, which are all the devices that work in remote positions, that have a certain degree of mobility or that simply have to be small and portable.

1.3 Edge-AI

Thanks to its versatility, AI has permeated a wide range of fields, moving from speech recognition to robotics, smart cameras and sensors, smart agriculture, smart wearable devices and AI assistants on board of smartphones.

However, all the tasks that require a certain degree of portability or mobility of the device itself have to come to terms with limited physical space and the inability, at least for a given amount of time, to access an external source of power. F.i., considering the robotics field, a mobile robot has a limited physical size, mostly occupied by the hardware that enables it to carry out its functions. The on-board computing has to come to term with the limited space as well, which means that also the computational power and memory available will be reduced. This is further exacerbated with smart cameras and sensors, whose space reduces to few centimetres. Furthermore, the progress of AI algorithms is currently associated with an increase in the requirements of memory and power capabilities, which goes in the opposite direction with respect to embedded mobile devices.

An interesting AI application involves deploying such techniques on *edge devices*, i.e., embedded devices often associated with limited computational power, memory and power consumption.

The set of techniques and tools that study how to bring complex AI algorithms to these edge devices is called *Edge-AI*.

Depending on the requirements of the target task, these kind of algorithms aim at reducing mainly: the *memory footprint*, in order to reduce the amount of bytes the AI algorithm requires to be stored in the device; the *inference time*, i.e. the amount of time it is required to obtain a prediction since the time the observation is provided or, in other words, the speed of the algorithm; and the power consumption of the tool, in order to allow the device to carry out its main task for as much time as possible.

In order to understand how big of an effort this work can be, it is necessary to think that each weight of which the network is composed of is usually stored in the `Float32` format, which takes four bytes. Considering that the most popular AI networks easily reach tens of billions of parameters, it means it can easily reach tens or even hundreds Gigabytes. Clearly, such an amount of memory is quite rarely available in embedded devices, which are typically associated with only few Kilobytes in

the most extreme cases.

Although it is not the scope of this work to bring such huge models on such extremely low-end devices, it is still necessary to put an effort to develop such algorithms while reducing the computational, memory, and energy requirements, in order to be able to actually deploy such techniques on such edge devices.

Several techniques exist to obtain AI algorithms that match the requirements imposed by embedded AI. We can identify two main strategies: reducing the memory footprint and improving computing performance for the target device.

Reducing the memory footprint is, as one can expect, a more general approach and can be applied to a wide range of AI algorithms. On the other hand, improving computing performance on a target device is a very specific and requires building libraries and tools that take advantage of the peculiarities of the embedded device to boost performance as much as possible.

1.4 Thesis Outline

This PhD thesis walks at the intersection of machine learning and control theory, studying applications to robotics in its many aspects. After covering some common background, this thesis is divided into two main parts.

The first part studies the perception of *robots*, focusing on applications based on deep learning techniques. In Chapter I, a cross-modal contrastive learning approach based on cGANs to transform sparse point clouds from mmWave sensors into depth images, preserving the distance information while producing a more comprehensible representation.

A first approach is proposed to translate sparse and noisy point clouds into depth images. It also provides an in-depth analysis of sparse point clouds from a radar sensor and produces a dataset labelling each point individually to enable per-point de-noising. Then, it sets the first steps for an investigation of the effects of network and connectivity to the control of real robots. In particular, a dataset is proposed to bridge a gap between robotics and communications, highlighting how controlling a robot should take into account the state of the connectivity and enabling machine learning techniques with data coming from both perception and network sides.

The second part deals with the *control* of mobile robots. It first proposes techniques to improve stability and performance of Deep Reinforcement Learning algorithms. These techniques are validated in a task that requires controlling a particular robot, known for its peculiar dynamics which makes the task particularly complex with classical and modern control techniques. A second contribution leverages Safe Reinforcement Learning techniques to control an industrial vehicle. In

this case, safety constraints are considered, bringing the machine learning based controller closer to a real world deployment and highlighting gaps of such tools with respect to real world applications.

Chapter 2

Background

The main purpose of this chapter is to provide the basics of all the topics that are involved in this work. A detailed description of the state of the art and related literature will be presented, instead, in the designated sections of the two parts of this work. Section 2.1 walks through the very basics of robotics, introducing concepts related to the perception and control of robotic systems. Section 2.2 introduces the main concepts of machine learning and reinforcement learning, which are used throughout the thesis. Finally, Section 2.3 introduces the context behind embedded devices and the necessity of developing dedicated tools to bring the powerful tools of machine learning to such devices.

2.1 Robotics

Robotics is the field of engineering that deals with the design and control of machines to solve given tasks with a certain degree of autonomy. The idea in robotics is to refer the most repetitive and dangerous tasks to machines (i.e. robots) so that humans could work in safe conditions and dedicate themselves to more intelligent tasks.

A first distinction in machines can be made by observing their mobility: if the machine can only operate in a fixed spot in a contained environment, it will be associated to techniques from conventional robotics. On the other hand, when the robot is actually capable of moving in a space, bringing its services wherever they are needed, it will be associated with the tools and techniques from *mobile robotics*.

Aside from the main, and intuitive, distinction, these two fields of robotics are further separated by other differences, even though as time goes by these differences grow thinner and thinner.

The first difference concerns the workspace in which the two types of tools work. For robots

working in a fixed position, typically named *industrial robots*, the workspace is fully known a priori, allowing the planning of the operations beforehand. In this scenario, the robot, typically a robotic arm, can move in the near surroundings to carry out tasks such as pick and place of objects, manipulation and selection of boxes and so on.

Mobile robots, on the other hand, can navigate in an environment that might not be known in advance. Moreover, these types of robot can work in an environment shared with other machinery or humans, whose position is not fixed, effectively becoming dynamic obstacles. This means that the robot must be equipped with the technology to perceive the surroundings, identifying the objects in the environment, both static and dynamic.

Another notable difference stems from the planning phase: with robotic arms, the plan to be executed is typically calculated once, before executing it, without the need for replanning. On the other hand, mobile robots need to re-compute the plan quite often during its operations. This difference comes from the workspace of the two: while for robotic arms the workspace is fixed, known a priori and often isolated from external tools or operators, mobile robots have to deal with bigger environments, often occupied by other objects and operators that might move, changing the conditions of the environment during time. This makes planning an activity that has to be carried out periodically, to not only prevent accidents and dangerous situations, but also to effectively carry out the task, especially in the case the objective moves or changes position.

In general, identifying obstacles and goals is not enough: it is necessary to design a tool that is able to leverage such information and compute control inputs to the robot in order to avoid collisions and timely reach the goal.

Then, we can distinguish between two main branches of robotics: *Perception* and *Control*.
The former deals with the elaboration of data from sensors. It is the research field that studies how to employ the raw information of sensors to obtain a useful representation of the surroundings. It also leverages sensors for proprioception, i.e. to get an estimate of the state (position, velocity, acceleration, etc.) of the robot being controlled. It provides all the information to act accordingly to the task and the evolution of the environment. In a control system, sensors enable the correction of errors and the computation of vital control action that would otherwise result in dangerous situations (collisions with obstacles, humans hurting, etc.).
The latter, instead, deals with controlling the robot, i.e. the actuation of motors and all the devices that enable the movement of the machine to reach a desired state.

Clearly, both branches are essential in a robotic system. In fact, only the cooperation between these two elements can make a robot *fully autonomous*, i.e. capable of performing a task without the intervention of an operator or a supervisor.

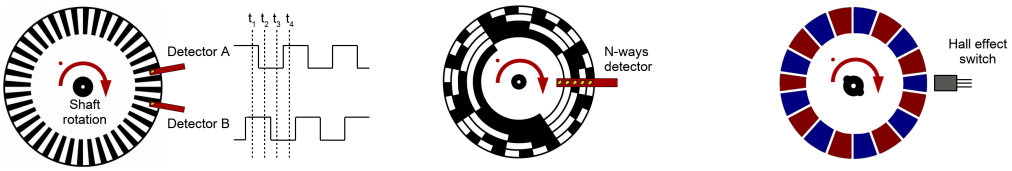


Figure 2.1.1: Types of encoders

Usually, the control systems that only leverage the control algorithm are called *open loop*, since the computation of the control input is performed without taking into account the evolution of neither the system nor the environment. In *closed loop* control techniques, instead, the controller leverages the information coming from the sensors to adjust the new control input to be computed as the system and environment evolve in time.

In this context, the perception of the robot enables identifying targets and obstacles, while the control algorithms are in charge of finding and actuating a plan to accomplish the task. Thus, it is necessary to develop perception techniques that can bring information from sensors in a form that can be leveraged by control algorithms.

2.1.1 Perception

Sensors are devices used to obtain an estimate of a physical phenomenon. They can be classified by the place where the phenomenon happens: when the phenomenon to be measured is part of the robot itself, then the robot will need a *proprioceptive* sensor; when the phenomenon to be observed is part of the environment and, in general, is outside of the robot's physical boundaries, then, it will need *exteroceptive* sensors.

Proprioceptive sensors

These sensors are fundamental in estimating the "internal" state of the robot, i.e. information such as the speed of the motors, the velocities and accelerations of the body of the robot, the positions (both relative and absolute) of the robot's joints, etc.

The sensors estimating the position of the wheels are called *encoders*. These sensors provide information about the linear (or angular) position of the wheel in the form of a digital code, by leveraging transducers (incremental optical encoders, Figure 2.1.1 [7] left), concentric disc with related transducers (absolute optical encoders, Figure 2.1.1 [7], center), or magnetic poles (Hall-effect encoders, Figure 2.1.1 [7] right).

Regarding the estimation of velocities and accelerations, the most common sensor is the *Inertial Measurement Unit* (IMU). This type of sensors combine accelerometers and gyroscopes to provide

an estimation of accelerations along the three axes and the orientation. IMUs are often employed to get an estimation of the position of the robot in the environment, by integrating the measurements over time. Accelerometers are often associated with good acceleration estimates at high frequencies, but when used alone they often lead to an error in the estimation that increases over time. On the other hand, gyroscopes provide a very accurate estimation when the dealing with slow dynamics but easily accumulate errors as the dynamics pace increases.

In order to provide accurate position estimates, both types of data are fused together with several approaches. The most simple one is a *Complementary Filter*: it uses a tunable parameter α to balance how much data are used from the two sensors to get the position (and orientation) estimate.

$$\text{orientation} = \alpha \times \text{gyroscope} + (1 - \alpha) \times \text{accelerometer}$$

Note that despite its simplicity, this filter obtains good performance for most applications. However, there exist more advanced technique to fuse these two sources. For instance, the Kalman filter [8] is a more advanced *sensor fusion* algorithm that leverages a mathematical model of the system to update the position estimate over time.

A recent approach to sensor fusion that enables the employment of more complex data such as images and point cloud leverages Artificial Neural Networks. However, these approaches are much more complex and, especially in systems with a real-time requirement, they become more difficult to be used efficiently.

Exteroceptive sensors

Exteroceptive sensors are all those devices that provide a description of the objects and the environment around the robot. The information these devices provide can come in many forms, from images to point clouds. For instance, digital cameras recreate an image (a three dimensional matrix with each dimension associated to a colour channel, usually Red Green and Blue, RGB) of the surroundings. Other types of images can be retrieved with other type of cameras, like infrared cameras or hyperspectral cameras. In these cases, the channels provided vary depending from the sensor.

Devices like LiDARs and radars, instead, provide a reconstruction of the environment in the form of point cloud, i.e. a list of coordinates that describe the distance of the detected object to the sensor. Also in this case the field of view depends on the sensor.

Time of Flight (ToF) sensors are composed of a light emitter and a receiver that detects the reflection of the emitted burst of light. The time it takes for the reflection to be detected provides an indication of the distance between the sensor and the object that reflected the ray.

Light Detection And Ranging (LiDAR) sensors are devices that, just like ToF sensors, emit a laser burst and wait for the reflection. 2D LiDARs create a planar reconstruction of the environment by emitting the ray, detecting the reflection and spinning around a fixed point. The data provided is a list of points, and the number of points depends on the number of samples taken in a full revolution, i.e. the resolution of the LiDAR. Multiplanar LiDARs follow the same working principle of 2D LiDARs, but adopt an array of ToF sensors to scan a 3D area. Another type of LiDAR adopts a 2D array of ToF to reconstruct a cone of distances.

There also is a particular configuration of cameras which can reproduce a 3D reconstruction of the environment. These are *Stereo cameras*. This type of devices actually use two cameras at a fixed distance (called *baseline*). Then, by leveraging the information of the two sensors, it is possible to estimate the distance of each captured object, similarly to the human vision system. This way, stereo cameras can produce, alongside with *left* and *right* RGB images, an estimation of the distances for each pixel, which can be represented both as a *depth image* (an single-channel image representing polar distances) or as a point cloud of distances.

Despite both proprio- and exteroceptive sensors providing useful information for both the robot's state and the surroundings, it is necessary to note that most classical and modern control approaches cannot directly leverage the most part of these type of information. Then, it becomes essential to develop algorithms to further analyse these information and extract the data necessary to describe the state of the robot and possible constraints. Clearly, the shape in which these information should be put depends on the control algorithm.

It is in this landscape that algorithms from, f.i., computer vision become essential to the control ones. Also, it is important to note that, alongside algorithms to "translate" raw data from sensors to useful information, many algorithms for *sensor fusion* have gained interest, in order to obtain robot's and environment's information which are more accurate and robust to noise and disturbances.

Then, in a control system dealing with robots in a dynamic environment, engineers can leverage a chain of three main components to effectively tackle the task. Starting from the sensors, data are analysed and information are extracted, then, the control algorithm computes the optimal control action based on the information received. This process, well-known to the control community as *control loop*, is represented in Figure 2.1.2.

2.1.2 Control

The role of control algorithms is make sure that the system carries out the task as intended. To do so, control theory divides the problem into *trajectory planning* and *trajectory tracking*. The role of trajectory planning is to compute a *trajectory* or a *plan* that, when followed, walks the robot

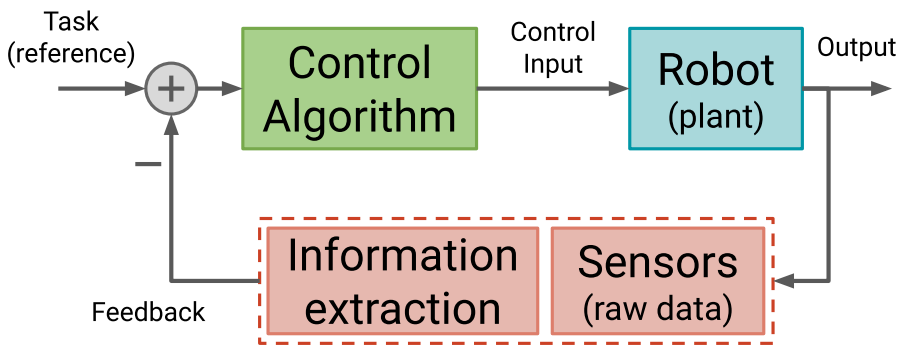


Figure 2.1.2: Control Loop

throughout the task. A trajectory is, then, a sequence of joints' position, motor speed, steering angles and values for all the actuated parts that compose the robotic system.

Then, given the trajectory, the trajectory tracking algorithm has to compute the *optimal* control inputs, i.e., the control inputs that actually bring the physical system to follow the desired trajectory. A common practice is to express the trajectory as an error function to the desired position. Given such a function, the role of the trajectory tracking is to minimize the error in the shortest amount of time. To do so, however, the control algorithm has to know how the system will react to a given action, i.e. it has to take into account of either the *dynamical* or the *kinematical* model of the system.

The dynamical model enables the computation of all possible movements of a robot while taking into account external forces and disturbances. Despite providing the most complete description of the robot, studying the dynamics of the system can be very challenging and keen to errors, since it requires the exact knowledge of the moment of inertia as well as the modelling of both static and dynamic friction. However, it is important to note that most of the time, a control algorithm only needs to know how the robot moves regardless of external forces, since these forces can be counter-balanced by the algorithm itself.

The kinematics of the robot describes how it can move without violating its own physical constraints. In other words, the kinematic model describes how the control input affects the velocity of the robot, neglecting the effects of external forces. Since it is simpler to derive than the dynamical model, most controllers are designed based on the kinematics of the system.

For example, for a robot moving on a plane, whose pose is described by the tuple $p = (x, y, \theta)$,

controlled by linear and angular velocities v and ω, then the following relationship holds:

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = \omega$$

The goal of the control algorithm is to compute the necessary control inputs to bring the robot from its current pose p to the desired one $p_d = (x_d, y_d, \theta_d)$. We can, then, define an error function $e(p_d, p)$:

$$e(p_d, p) = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} x_d - x \\ y_d - y \\ \theta_d - \theta \end{bmatrix}$$

from which, a very basic controller can be derived by simply computing the linear and angular velocities as:

$$v = k_1 e_x$$
$$\omega = k_2 e_\theta + k_3 e_y$$

where k_1, k_2 and k_3 are controller gains.
The one just illustrated is very basic linear controller that yields good performance for very simple use cases but falls short as system requirements become more demanding.

A very popular controller, which strikes a trade-off between simplicity and efficacy, is the Proportional-Integrative-Derivative (PID) controller. As its name suggests, it uses three components to compute an effective control action: 1) the *proportional* $k_P e(t)$; 2) the *integral* $K_I \int_0^t e(\tau)\mathrm{d}\tau$, which acts as an accumulator of the error in order to address small errors too; and the *derivative* $K_\mathrm{d} \frac{\mathrm{d}e(t)}{\mathrm{d}t}$, that evaluates the variation in time of the error to prevent abrupt control actions.

Despite its great performance in linear systems, the PID can only be applied to a very small subset of non-linear systems. In the field of control theory there are non-linear controllers which are able to deal with system non-linearities and more advanced approaches, such as Model Predictive Control [9], which take advantage of the dynamical model of the system to compute the control actions that satisfy the system's constraints.

In classical and modern control theory, the design process of a controller requires: 1) the identification of the kinematic (or dynamic) model of the system; 2) the design of the planner to

produce effective trajectories; 3) developing the set of tools that, given raw data from the sensors, produces useful information for the controller; 4) and, finally, designing a controller which takes into account the peculiarities of the system to be controlled.

This chain of tools has proved to reach high performance, however, it comes with its downsides: the design process is particularly time consuming; the whole chain of tools can require a long execution time; it tends to accumulate errors throughout the chain. Furthermore, for complex environments and tasks it may not be enough or may not meet the requirements.

To solve these problems, new approaches arise to leverage tools from the machine learning field in order to map the raw data from the sensors to the control input. These algorithms can be designed to also take into account the constraints and dynamics of the system, without necessarily requiring their formal description. Furthermore, since these tools map the data directly to the control inputs, there is no need to develop neither algorithms to extract information from raw data nor trajectory planning algorithms.

2.2 Machine Learning

Artificial Intelligence and Machine Learning are terms often used interchangeably, but they actually refer to two distinct class of algorithms. With AI we refer to the algorithms and tools that leverage a set of rules, known a priori, to solve a specific task. Despite being known for high performance, these approaches are quite limited to specific task and it becomes difficult to adapt such techniques to a task (even slightly) different from the one the tool is designed for.

Machine Learning is the set of algorithms that strives to solve a problem by learning the relationships and the rules between the data observed and the desired output.

These kind of tools work by receiving in input some information and, after elaborating it, they produce an output, or *prediction*, whose form is strongly related to the task: in the context of vision and object recognition, one kind of output might be a label describing the objects in an image or a video; concerning language processing, the output can be a phrase or a text; when controlling a machine, the prediction will be some control actions to do its movement.

In general, all machine learning tools are designed to solve tasks by learning from experience, which is usually conveyed in the form of data. Therefore, collecting representative and expressive experience (i.e. data) becomes of utmost importance.

We can divide machine learning tools into two main parts: the *approximation function*, a structure which observes the input data, elaborates it and produces the output, and the set of rules used to improve the performance (i.e. learning algorithm) of the approximator based on the data available.

In order to be able to "learn", the approximator leverages *weights* to express the connection in the input data and how they are related to the desired output. Then, the goal of the sets of rules becomes to find new values for such weights in order to improve performance.

In the years, many researchers have proposed several approximator models and architectures, such as polynomials, tree-like structures and, more recently, neural networks.

The rules and tools to let the approximator learn strongly depend on the task. A first classification of machine learning tasks and, possibly, the most common, is: *Supervised* learning [10], *Unsupervised* learning [10–12], and *Reinforcement* learning [13].

Supervised learning refers to the machine learning techniques that associate each input data to a *label*, which represents the expected output that the approximator should produce. The learning process is "supervised" by a distance metric between the prediction and the related label.

Unsupervised machine learning algorithms are all those techniques that do not (and most times cannot) leverage labels to express the desired output with respect to the observed data. In this class of algorithms fall all those approaches that strive to identify patterns or clustering data under the same groups. A particular subclass of this type of algorithms is *Contrastive* learning [14, 15]. This approach exploits similarities (and differences) in the input data itself to drive the learning process. In particular, the goal is usually to compare the input data to some other data. The goal of the approximator is to estimate the distance of the input data to the data it is compared to. It is particularly employed in *representation* learning and in *auto-encoders* [16]. Concerning robotics, these approaches are commonly used to fuse data from different sensors, exploiting the *multi-modality* of approximators ([17, 18]).

Reinforcement learning is the framework under which fall all those techniques that are involved in the control of a process or execution of a task [13]. Much like unsupervised learning, these algorithms do not leverage any labelled data. The learning process is driven by a performance metric which, most times, measures the progress or performance in the target task. Usually, reinforcement learning do not exploit a dataset. In fact, the training process is composed of a phase in which the approximator interacts with the environment, collecting data in the form of *experience*. Each experience allows to associate the input data with the prediction (a *control action*) and the observed performance metric. The range of application of this framework is very wide, since it can be applied to any process to be controlled. In fact, it has been applied to games ([19–21]), to video-streaming platforms ([22–24]) and to robotics ([25–27]) too, as the controller of various types of robotic systems.

2.2.1 Artificial Neural Networks

This kind of approximation function is named after biological neurons, since it is meant to mimic the neurons in the human (and, more in general, mammal) brain. Like the biological brain, artificial neurons are connected with each other through weights, acting like synapses. Following the biological counterpart, artificial neurons are organized in a hierarchical structure, which is implemented through layers, each associated with an arbitrary number of neurons.

Then, an approximation function f with parameters θ can be defined as:

$$y = f_\theta(X)$$

with X and y being, respectively, the input information and the approximator's prediction. The size of the input data F represents the number of information (*features)* available to the approximator, while the number of features to be predicted $|y|$ is strictly related to the task: for the classification of the input data into n classes, one possible implementation is to let the network predict n probabilities, one for each class, and consider the class associated with the highest probability as the predicted one; in the task of controlling a unicycle robot, a possible approximator will predict the linear and angular velocities, i.e., $|y| = 2$.

Let the l-th and the $(l+1)$-th layers of the network be composed, respectively, of s^l and $s^{(l+1)}$ neurons. Then, the set of parameters identified by θ^l will be an $s^l \times s^{l+1}$ matrix.

The first layer ($l = 0$), represents the input data and the associated weights θ^0 will be a matrix of size $s^0 \times s^1 = F \times s^1$.

The $(L-1)$-th layer be associated with θ^L, a matrix of size $s^{L-1} \times s^L = s^{L-1} \times |y|$.

Note that, since after the L-th layer there is no further computation, the last layer of neurons (which are already the output values) is not associated with any matrix. Therefore, the number of neuron layers L will be associated with $L-1$ weight matrices.

All the layers in between the first and the last layers are called *hidden* layers. Moreover, the l-th layer represents a so-called *latent* representation. That is, a representation of the input data in a different (higher or smaller) space.

To further define a neural network architecture, let z^l be the output of the l-the layer. Then, $z^l = z^{l-1} \cdot \theta^l$. With $l = 0$: $z^0 = X \cdot \theta^0$.

Then, let B^l be the matrix of biases and f^l be the activation function at the l-th layer, we define a general form of the output at the l-th layer as:

$$a^l = f^l(a^l - 1 \cdot \theta^l + B^l)$$

Similarly, $a^0 = X$ and $a^L = y$.

So far we have described what's called a *feed-forward* neural network, i.e. an architecture whose output is obtained by feeding the i-th layer with the output of the previous one, from $l = 0$ to to $l = L$. In particular, the architecture presented is called *Multi-Layer Perceptron* (MLP, [4]), a neural network mainly composed of linear operations and activation functions. Its simplicity and effectiveness make it one of the most common types of architecture deployed. However, one of the main limits of such architecture is that it becomes incredibly demanding in terms of computational resources as the input size grows. For instance, it is hardly adaptable to data like images or graphs.

Along with the MLP, another feed forward neural network is the *Convolutional* Neural Network (CNN, [28]). This type of architecture applies the convolution operator to the input data, by leveraging a learnable kernel (i.e., a kernel with weights updated during the learning process to improve performance). The kernel is overlapped to the input data, typically an image, and moves across its size, applying the matrix multiplication between the kernel and the overlapped area. The convolution is often combined with a pooling operation, to select the values out of the matrix multiplication operation. This architecture provides high performance at extracting geometrical information from the input data and is mainly used to elaborate image-like information, i.e. data in the form of single- (or multi-) channel matrices.

Another interesting neural network architecture is the *Graph* Neural Network (GNN, [29, 30]). Its main advantage with respect to the previous architectures stands in its ability at elaborating unordered types of data, like lists of coordinates, or lists of points etc. This type of architecture leverages operations from the *graph theory*, such as message passing or neighbour evaluation.

Contrary to feed forward neural networks, Recurrent Neural Networks (RNNs, [31, 32]) are built allow storing information throughout predictions, enabling the elaboration of previously seen data and embedding in its structure the concept of temporal sequence. Among this type of architectures one can find Long Short Memory Term (LSTM, [31]) cells, or Gated Recurrent Units (GRUs, [32]).

Note that, just like CNNs, LSTMs and GRUs can be concatenated to other *types of layers*, obtaining more complex and, possibly, expressive architectures, in order to extract useful information from the input data under many different aspects.

Training neural networks

Training neural networks is the process under which the weights are updated in order to improve the overall performance. In this process, the performance metrics defines the error from the ideal result. Representing the neural network with its weights θ, we define the *cost function* $J(\theta)$.

Then, in order to minimize such a cost it is possible to compute its gradient with respect to the

network parameters $\nabla_\theta J(\theta) = \frac{\partial J}{\partial \theta}$.

Such a gradient can, then, be used to update the weights according to the steepest descent algorithm:

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial J}{\partial \theta}$$

where θ_t and θ_{t+1} represent the weights of the network before and after the update and α is the *learning rate*. Note that, while the gradient $\frac{\partial J}{\partial \theta}$ provides the direction towards which the weights should be moved to minimize the cost function, it does not provide any indication about the scale of such step. This means that its norm only depends on the slope of the cost function, creating two critical cases:

1) the slope is too small; this condition produces updates which are too small and would require a large amount of time to reach the cost function's minimum.

2) the slope is too big; this means that the updates are also too big, meaning that the performance would oscillate around the function's minimum or, in the worst case it would diverge.

The learning rate acts as a tunable hyper parameter, allowing to take steps towards the right direction with a "controlled" modulus.

In practical terms, however, $J(\theta)$ may not be known a priori, or may simply be too "general purpose". For example, in the problem of binary classification of cats and dogs images, a Binary Cross Entropy (BCE, [33]) might be used. However, despite BCE being well defined and known a priori, it does not represent the task. Hence, it is the combination of data, cost function and (possibly wrong) predictions, that are needed to accurately describe the task, creating the conditions under which a proper gradient can be computed.

Then, an important thing to understand is how much data is enough data. Clearly, the amount of data for the network to minimize the cost function mainly depends on the task complexity and the capabilities of the model. More complex architectures may be able to better learn the task, but usually require more data.

For example, in a classification problem, the data should represent all the possible cases (both simpler and more difficult or rare ones) to let the weights adapt to such conditions. It is fundamental, during training, to provide the right amount of data for every case the network should be able to get. Note that consequently, the gradient also depends on the data.

This means that using all the available data improves the quality of the updates. Then, using all the available data may seem a reasonable strategy to solve the optimization problem as fast as possible. However, this is not true. During the computation of the gradient, any noise in the input

data may strongly impair the learning process. That is, it may lead to suboptimal performance and local optima. This is due to many reasons.

Using the whole *dataset* makes the direction of the dataset strongly affected by the largest group of data representing the same scenario. This means that the update will not take into account scenarios poorly represented in the dataset, obtaining the so-called *overfitting*. From a practical perspective, using the whole dataset may require prohibitive computational resources and/or time, to get to a solution which may still not solve the overall task.

Then, fixed a dataset of $|X|$ data, a solution is to define a *batch* $\mathcal{B}$ of data with size BS. The batch is, then, used to compute the gradient on a smaller number of samples, obtaining an update which is influenced only by the data in the batch, allowing rarer samples to affect the updates too. Also, smaller batch sizes enable for faster computation a typically require less computational resources. However, reducing BS too much has a self-defeating effect: when BS is too small, the gradient will be computed on too few samples and will also be strongly influenced from all the data that might not properly express the task (or the cost function).

In the remaining of this section, a background on Reinforcement Learning and Deep Reinforcement Learning is provided.

2.2.2 Reinforcement Learning

Reinforcement Learning (RL) is a learning framework in which an *agent* has to carry out a task in a given *environment*. The information describing the current state of the environment is provided through *states*, while the *control actions* (or just actions) represent the mean through which the agent interacts with the environment, modifying its state. The agent can, then, be seen as a mapping function, much like an approximation function in the context of classical machine learning, that maps the current state to actions (or control inputs).

The learning process requires the agent to interact with the environment, collecting experience (i.e., data) based on which the training is performed. The amount of data required to train agent is related to the concept of *sample efficiency*: the less data is required, the more the algorithm is efficient. Note that this is not just a matter of efficiency: interacting with the environment might be costly, both in terms of time, computational resources and equipment required. Also, especially in the initial steps of learning, the agent may put to risk the equipment and, possibly, the human operators involved as well.

The cost function driving the learning, in this case, is strongly related to the task and may represents the desired high level performance. The cost function may also represent a reward, i.e. a positive value associated to a condition (or course of actions) that lead to desired performance. For this

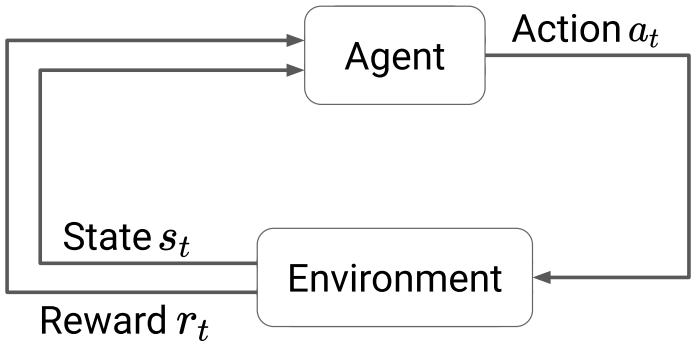


Figure 2.2.1: Interaction between agent and environment in a Markov Decision Process

reasons, in RL, most of the times, such function is referred to as "reward function".

In RL, the learning process is modelled as Markov Decision Process (MDP) (Figure 2.2.1). The MDP framework is defined as a tuple composed of:

- the set $\mathcal{S}$ of all possible states s;
- the set $\mathcal{A}$ of all possible actions a;
- the *transition function* $T(s, a, s')$, which describes the probability with which, applying action a in state s brings to the state s', i.e., $P(s'|s, a)$;
- the reward function $R_t = R(s, a)$, with $R_t \in \mathbb{R}$.

By leveraging the MDP framework, we make the assumption that the process abides to the Markov property:

$$P\left(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \dots\right) = P\left(s_{t+1} \mid s_t, a_t\right)$$

which translates to an environment whose evolution is only related to the current state and the action taken in such state. In other words, this assumption relieves the agent from keeping track of the history of the evolution of the environment, which simplifies the problem also from a computational point of view.

At each step of interaction, the reward acts as a *signal* of the performance of the agent. Then, the main goal of the agent is to maximize the total amount of rewards it can receive. Note that maximizing only the current step's reward may not be enough. For example, while solving a maze, the agent may find itself in a close position to the centre of the maze, but blocked by some walls to prevent it from actually reaching the solution (*local optima*). A good strategy would make the agent take some steps away from such position in order to actually reach the maze's centre, but this can also lead to intermediate positions which are farther from the centre, resulting in temporarily

poorer reward signals. However, such steps are necessary to reach the actual solution of the maze, which is the whole purpose of the task.

Therefore, the purpose of the agent can be formalized as the maximization of the *expected return* G_t: a function of the sequence of rewards received. A simple function may be the sum of the reward signals:

$$G_t \doteq R_{t+1} + R_{t+2} + \cdots + R_T$$

with T being the last agent-environment interaction step. In this formulation, we are assuming that the interaction process has a condition (time, victory, loss...) under which the process terminates. In other words, we are assuming the interaction process is *episodic*, where each *episode* starts with an *initial* state and ends with a *terminal* state. This means that, after the terminal state, the process is reset, and a new initial state is provided. How the initial state is computed is strictly dependent on the task and may be sampled from the distribution of initial states. However, not all processes are *episodic*. In fact, there do exist *continuing* tasks, in which the agent interacts with the environment for a very long time (e.g., robotics). In this case, the expected return can be reformulated as:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

with $\gamma : 0 \leq \gamma \leq 1$ being a parameter, named *discount rate*. When computing the current expected return, future rewards will be scaled by a γ^k factor, which helps defining how much the agent should care for future reward signals. When γ tends to 1, the future rewards are considered just like the current one, and the sum is bounded only if the reward function is bounded. $\gamma = 0$ nullifies future rewards and the expected return is only influenced by the current reward signal, neglecting future rewards.

However, computing such expected reward is practically not possible, as one is required to know future rewards. For this reason, many RL algorithms employ a *state-value function*, a function that estimates how good a given state is, considering the expected returns as well. Expressing the *policy* π as the one the agent is currently employing, then π expresses the probability of taking action a when observing the state s, i.e., $\pi(a|s)$. Then, the value function for the state s when following the policy π is defined as:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid s_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid s_t = s \right], \text{ for all } s \in \mathcal{S}$$

where $\mathbb{E}_\pi$ represents the expected value of the return when following π.

Analogously, it is possible to define the *state-action value function*, a value function that estimates

the expected return when being in state s and taking action a:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi \left[G_t \mid s_t = s, a_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid s_t = s, a_t = a \right].$$

Then, by applying Monte Carlo methods, it is possible to estimate such value functions from experience collected during the interaction process [13]. Note that these functions can be defined recursively by using the Bellman equation [34]:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi \left[G_t \mid s_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid s_t = s \right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right] \\ &= \sum_a \pi(a \mid s) \sum_{s', r} p\left(s', r \mid s, a\right) \left[r + \gamma v_\pi\left(s'\right) \right], \qquad \text{for all } s \in \mathcal{S} \end{aligned}$$

It is worth noting that, in this formulation, the terms $\pi(a \mid s)$ and $p\left(s', r \mid s, a\right)$ act as weights for each action, effectively averaging over all possible actions.

We can, now, reframe the purpose of RL algorithms into a search for an *optimal policy*, i.e. a policy associated to the "best" value function. In general, a policy π is better or equal to a policy π' if and only if the inequation $v_\pi(s) \geq v_{\pi'}$ for all $s \in \mathcal{S}$. Then, the optimal policy π_* is that policy associated with the value function which is greater or equal to all other policies' value functions:

$$v_*(s) \doteq \max_\pi v_\pi(s)$$

Note that despite the value function being possibly associated to multiple policies, all the policies that produce the best value function are indicated with π_*. Also, the best policies will be associated with the best state-action value function q_*:

$$q_*(s,a) \doteq max_\pi \; q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*\left(s_{t+1}\right) \mid s_t = s, a_t = a \right]$$

It is worth noting that the tools presented so far are but means to evaluate each policy. In order to actually employ such policy, a possible approach is to evaluate each action-state value function

for a given state s and take the action associated to the best value:

$$\begin{aligned}\pi'(s) &\doteq \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi\left(s_{t+1}\right) \mid s_t = s, a_t = a\right] \\ &= \arg\max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi(s')\right]\end{aligned}$$

The resulting policy is called *greedy policy*, since given a policy π, when observing state s it will always take the same action.

Notice that, this process of selecting the best action from a policy π results in new policy π' which actually improves performance, giving birth to the *policy improvement* process. The algorithm that takes advantage of such process iteratively is called *Policy Iteration*.

Today, in the field of Deep Reinforcement Learning (DRL), the policy and the functions v (or q) are typically approximated by neural networks.

There are several types of DRL algorithms, which can be grouped by several factors.
Value-based vs Policy-based: Value-based algorithms [20, 35] will parametrize the value function and the action can be sampled by selecting the one that maximizes $Q(s, a)$. Policy-based algorithms [25, 36–38], instead, will parametrize the policy without explicitly representing the value function.
Actor-Critic methods: these methods try to explicitly learn both the policy (actor) and the (action-)state value function (critic), taking advantage the positive aspects of both the previous methods [25, 36–38].
On-policy vs Off-policy. The on-policy algorithms [25, 36] follow an update rule which strictly correlates the experience to the policy that collected it. Off-policy algorithms [37, 38] can, instead, take advantage of the experience collected, regardless of which policy was used to obtain it. These algorithms are often (but not necessarily) associated with improved *sample efficiency*, i.e. less data required to learn a good policy.

2.2.3 Safe Reinforcement Learning

The class of DRL algorithms introduced so far requires interacting with the environment and exploring the action space to derive an effective policy. However, this process of trial and error may be dangerous both for the equipment and humans involved, and completely prohibitive in case where the process to be controlled directly affects the health of human beings (e.g. healthcare applications). In general, the process to be controlled may come with *constraints*, i.e. conditions that should not be violated. However, the methodologies introduced so far do not address any constraints, both at

training and deployment time, and, at the most, they can add components in the reward function to express some form of cost, but they clearly are not able to guarantee such constraints.

For these reasons, *Safe Reinforcement Learning* (SRL) techniques adopt the *Constrained Markov Decision Process* (CMDP) as the building framework. With respect to the MDP, this framework embeds the set of costs $\mathcal{C}$, which represents all costs $\{c_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}, i = 1, 2..., |\mathcal{C}|\}$. This expanded formulation enables algorithms to effectively deal with constraints.

2.3 Smart Embedded Devices

Over the years, neural networks have significantly advanced the capabilities of artificial intelligence in several domains. This progress has driven a growing interest in deploying these models on embedded devices, like smartphones, ground robots, drones, medical devices, and other embedded systems. However, such capabilities are practically obtained through a set of parameters that embody the neural network. Consequently, these models require in general substantial memory capacity to store these parameters, which are typically represented in full precision (32 bits or more per value in formats like float or double). Moreover, DNNs demand considerable computational resources both during training, usually conducted on high-performance, power-intensive CUDA-compatible GPUs, and during inference, where the network is effectively deployed.

In order to make neural networks deployable on such devices, several techniques have been proposed to reduce both the size of these models and their inference time, as well as the complexity of these operations, in order to account for the limited storage, battery life, and computational capabilities of embedded devices. These techniques become essential when dealing with such devices. A common approach to reduce the memory footprint of neural networks is *quantization*. Quantization is a technique that decreases the number of bits used to represent network parameters in memory while containing the loss of accuracy and preserving the network's expressiveness as much as possible.

However, this process is non-trivial since rounding or truncating their values to a lower-resolution representation can compromise the performance, leading to useless sets of weights. Nonetheless, such approaches remain effective: this is easily explainable by noticing that the quantization operations can be seen as noise applied to the weights. Nevertheless, neural networks are inherently robust to such noise thanks to their ability to generalize and produce similar outputs given similar inputs. This robustness stems from their design, which often involves overparameterization, redundancy, and the capacity to approximate complex functions despite small perturbations.

Clearly, this holds for quantization to a number of bits which is close to the original one. The fewer the bits, the farther the performance will be driven from the original network.

Clearly, depending on the application, one may prefer to still quantize the network to 3 or 2 bits to be able to deploy the model, despite dramatically lowering performance.

In this context, we can identify two quantization methodologies: *Post-Training Quantization* and *Quantization-Aware Training*.

2.3.1 Post-Training Quantization

Post-Training Quantization (PTQ, [39]) involves applying quantization to a fully trained model without requiring additional training or significant computational effort. Typically, it reduces the precision of the parameters (weights and biases) from 32-bit floating-point to 8-bit integers or other lower-precision formats.

An important aspect of PTQ is related to the quantization of activation functions. In fact, the quantization typically maps the real-valued outputs of the activation function into a finite set of discrete values. This requires the computation the *scale* and the *zero-point* of the function. Computing such values strongly affects the performance of the resulting function.

Hence, we can identify two main approaches that specifically deal with the activation functions.
Static Quantization [39]: in order to compute the scale and the zero-point, a smaller dataset is used to carefully calibrate such values. Then, a clipping operation is applied to bound the output and prevent outliers from rare or extreme values.
Dynamic Quantization [39]: in this case, activation functions are quantized on-the-fly during inference, using runtime statistics instead of a pre-collected calibration dataset. In general, this is a more flexible approach, but can be less accurate than static quantization since it may not capture the data distribution as effectively.

2.3.2 Quantization-Aware Training

The goal of Quantization-Aware Training (QAT, [39]) is to train an already quantized neural network. The idea is to leverage the learning process to better exploit the lower bit representation and effectively reduce the performance loss. In particular, the inference process during training is carried out with a quantized (weights, biases and activation functions) neural network, while the gradient is computed in full precision. This is necessary to prevent any information loss on the gradient that may lead to updates that do not reflect the computed updates. In fact, quantizing the gradient would mean actively changing such values, nullifying the gradient computation and possibly leading to divergence.

The main advantage of QAT is that, by quantizing the network at training time, it forces the learning process itself to find an effective lower bit representation, instead of trying to adapt a

full precision neural network to a set of lower precision weights. However, QAT requires more computational resources and longer training times compared to PTQ, which can be a problem in scenarios where computational efficiency is crucial, or when the training process is too long and the model is already available.

Part I

Perception

Chapter 3

Point2Depth: a GAN-based Contrastive Learning Approach for mmWave Point Clouds to Depth Images Transformation

3.1 Introduction

Sensors working in the ultraviolet, visible, and near infrared light are extensively used in mobile robots for the perception of the surroundings. These sensors offer many advantages, including high fidelity, precision, interpretability, and ease-of-use. The most commonly used sensors in this category are RGB and RGB-Depth (RGB-D) cameras and LiDARs, which enable obstacle detection, obstacle avoidance and Simultaneous Mapping and Localization (SLAM). These types of sensors are widely used in the literature and there exist many approaches which leverage their data to effectively tackle various mobile robotics tasks. However, visible light based sensor, can fail to provide reliable data in adverse environmental conditions, such as in the case of the presence of smoke, fog, and occlusions. On the other end, mmWave radar sensors are not affected by such conditions and can provide information of partially or fully obstructed obstacles and of objects made of materials that hardly reflect visible light (f.i., glass), which are often invisible to LiDARs. While mmWave sensors overcome and are more affordable, they are unfortunately more susceptible to noise and produce sparser PCs than LiDARs'. In particular, single-chip radars, even at a millimeter wave lengths, have a much lower azimuth resolution compared to LiDARs, resulting in point clouds with lower resolution. This limits their use to basic collision avoidance applications, while more advanced applications might require larger and more expensive mechanical radars that might not be suitable to most mobile robotics applications.

To address the limitations of PCs generated by mmWave radars, several approaches have been proposed in the literature for data processing and interpretation. Some authors have focused on denoising and interpolation methods, such as Kalman filtering and Gaussian Process Implicit Surfaces, to improve the accuracy of the point clouds [40]. Others have proposed feature extraction and classification algorithms to identify objects from the point clouds ([41, 42]).

Recently, there has been growing interest in using deep learning techniques, such as Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs), to improve the interpretation of mmWave data. In particular, cGANs have been applied for various tasks, such as object reconstruction and semantic segmentation [43, 44].

In this chapter, a cross-modal contrastive learning [45], [42] approach based on cGANs [46] is proposed to transform mmWave point clouds into depth images [47]. To this end, an extensive data acquisition has been conducted to create a multi-modal dataset in which each point cloud is strictly related to a depth image. The mmWave PC is rototranslated into the camera reference frame. Furthermore, for each PC, we only retain points in the field of view of the camera, since points outside the field of view cannot be compared to any information in depth image. The goal is to leverage the reliability of the mmWave sensor while mitigating the sparseness of its data.

3.2 Related work

3.2.1 Contrastive Learning

Contrastive learning [45] is a method in which a model is trained to identify patterns in the input data , while repelling dissimilar ones. This framework improves performance in both supervised and unsupervised contexts, and has also been used as a "pre-training" technique [48]. Previous studies (such as [49]) have demonstrated that several types of losses can be employed to extract useful information. In [50], contrastive learning was effectively used to extract meaningful representations from high-dimensional data (i.e., images) that were then fed to a Reinforcement Learning (RL) agent. The approach resulted in state-of-art performance in many visual tasks from Deep Mind control suite and Atari games.

In this work, we aim to leverage contrastive learning as a training technique to derive a cross-modal representations from mmWave PCs to depth images. This process enforces a comparable latent space between the two types of data.

3.2.2 Conditional Generative Adversarial Networks

Generative Adversarial Networks [51] were first introduced as a training framework for generative models. The GAN framework consists of two models: a *Generator* $\mathcal{G}$ and a *Discriminator* $\mathcal{D}$, in competition with each other. The generator's goal is to produce a faithful duplicate of a target signal from a noise signal, while discriminator's goal is to differentiate between signals produced by the generator and true signals from the given dataset. This creates a dynamic in which the generator tries to deceive the discriminator into believing its signal is a true signal, while the discriminator corrects the generator by accurately distinguishing between true and falsified signals. However, the designer has no control over the generation of fake signals in this setting.

In [46], the authors expanded on this work to introduce control over the way the generator outputs signals, resulting in the development of cGANs. In [52] a cGAN with an autoencoder-like generator is used to learn a mapping from an input image to an output image.

In this chapter, a cGAN model incorporating an autoencoder-like ([53]) generator is employed to map an input colour image to an output depth one. This results in the synthesis of an informative latent representation.

3.2.3 Point Cloud elaboration

Point Cloud elaboration is an open issue in literature. Unlike other types of data (f.i. images), PCs are unordered data, meaning that a particular value has the same informative value independently of its position in the data structure. This property makes it difficult to apply traditional approaches that rely on position information, such as f.i., convolutional and pooling layers. In [41], authors propose a methodology to extract both local and global features through the use of a *transformation network* (T-Net). The proposed technique has been shown to be effective for the classification and segmentation of large PCs. [42] proposes a cross-modal contrastive learning approach to learn a LiDAR-like latent representation which is used for semantic labeling in an occupancy grid and as input for an RL agent for autonomous navigation.

In this chapter, we follow [52] and apply a deep learning approach to derive a latent representation of *sparse* PC to effectively convert the input PC into a depth image.

3.2.4 RGB to Depth

In [54], the authors utilize a sparse depth map and the associated RGB image to solve the depth scale ambiguity. The results show high accuracy compared to the original depth image. [55] presents a model architecture that advances the state-of-art performance, making use of an encoder that

extracts multi-scale features, which are given in input to the decoder, making use of skip-connections.

In this chapter, a deep neural network is employed to generate a latent representation of the input image, which encodes all the necessary information about the distance. It is important to note that, as only the encoder part of the autoencoder will be employed in the final model, skip-connections cannot be utilized.

3.3 mmWave Point Clouds to Depth Images

In this section we describe the methodology adopted to transform sparse mmWave PCs into depth images.

First, we observe that PCs and depth images share, to some extents, the same kind of information, which is the distance between an object and the sensor. However, if on one hand the depth image is usually produced by leveraging two RGB cameras and the related physical information, e.g. displacement between sensors, focal distance, field of view and so on, on the other hand the mmWave relies on its physical principles to produce PCs.

We also note that, when both sensors share the same point of view, mmWave PCs can be considered as a sparser representation of the depth image produced by an RGB-D camera. However, mmWave are not impacted by adverse light and environmental conditions, making them a more reliable source of depth information. Further, due to the different principles on which the two sensors are based, the resulting data cannot be considered as one the down-sampling of the other: two different scenes will produce two distinct results for both sensors.

We present *Point2Depth*, a model designed to tackle the sparsity of mmWave PCs, consisting of two components as shown in Figure 3.3.1: (1) a PC Encoder Neural Network (NN), *Point2Latent*, that takes mmWave PCs in input and generates a *latent* representation; (2) *Latent2Depth*, a decoder NN which converts the *latent* representation into a depth image.

As shown in Figure 3.3.2, this decoder is part of a cGAN, namely *RGB2Depth*, that is trained to generate faithful depth images from RGB images while encoding the initial information into a latent space. To ensure the produced latent representation is useful, the Point2Latent Encoder is trained using the contrastive learning paradigm. The overall training scheme is summarized in Figure 3.3.3.

Point2Latent is trained to produce a latent space that is as close as possible to the one produced by RGB2Depth, when observing the mmWave PC. This allows Point2Latent to translate the input PC into a depth image-like latent representation. The Latent2Depth then decodes this representation into a depth image, in a decoupled process.

The remaining of this Section is divided into two parts: Section 3.3.1 provides a detailed

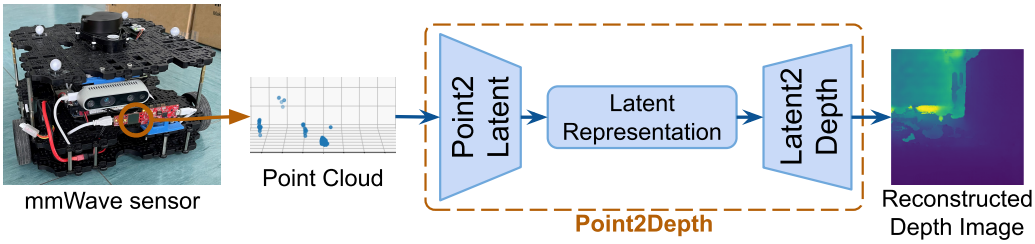


Figure 3.3.1: Point2Depth Architecture

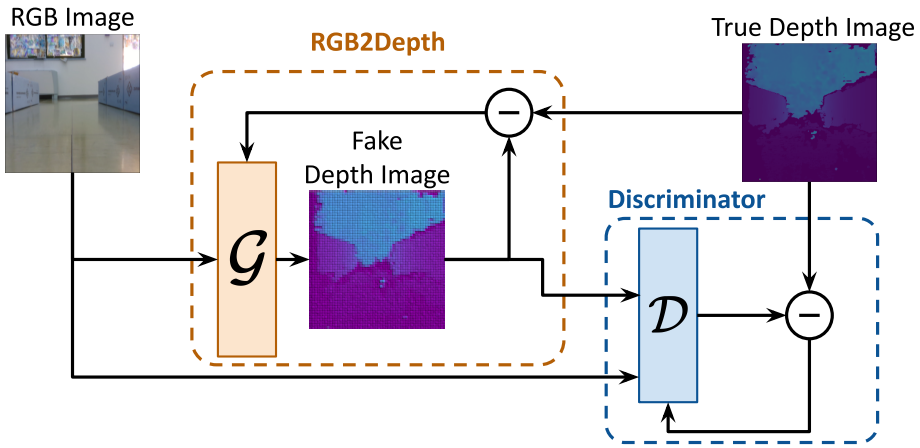


Figure 3.3.2: RGB2Depth training scheme

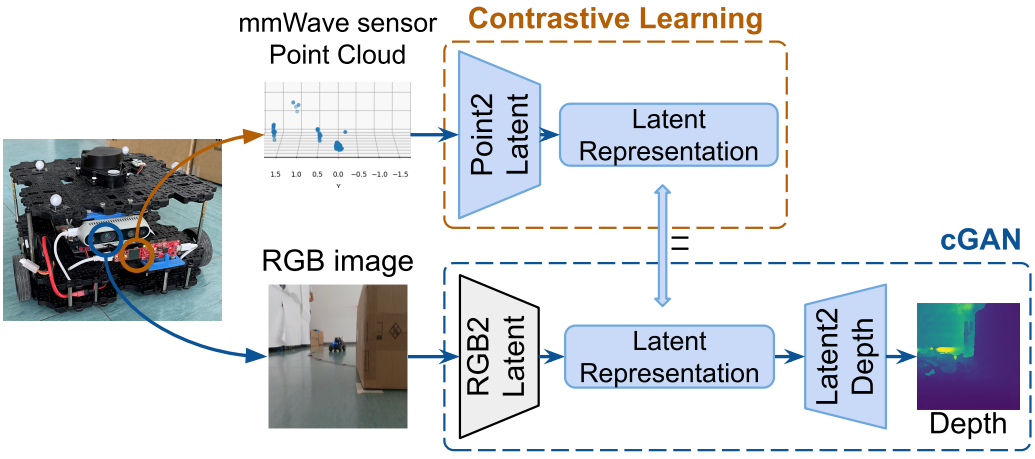


Figure 3.3.3: Point2Depth Overall Training Scheme

description of the approach used to train the depth image decoder; Section 3.3.2 explains the training of the PC encoder to generate a useful latent representation.

3.3.1 RGB to Depth Image cGAN

We now describe the training scheme employed for the component *RGB2Depth* used to convert RGB images to a depth image. Figure 3.3.2 shows that this component is trained as a cGAN with a generator structure that can be viewed as an auto-encoder. RGB2Depth takes as input a color image rather than the corresponding depth image, which could lead to an encoded latent space optimized for image reconstruction instead of conveying effective distance information. This approach, while seemingly counter-intuitive, has the merit of encouraging the neural network to extract meaningful features from the input data, resulting in a more informative latent representation. Additionally, this approach reduces the risk of overfitting to the training dataset.

Let $\mathcal{G}$ and $\mathcal{D}$ identify the *generator* and the *discriminator*, respectively. The training dataset is defined by the couples $X^i = \langle X^i_{RGB}, X^i_D \rangle, i = \{0, ..., |X|\}$ with $|X|$ being the dataset's size. Note that X^i_{RGB} identifies the i-th color image, X^i_D represents the depth image associated to the i-th color one.

Discriminator $\mathcal{D}$

In our approach, $\mathcal{D}$ is trained to observe both an RGB image and its associated depth one, and to determine whether the input depth image is a genuine or fake one. During the training process, the following loss function is optimized over a batch b of data:

$$\mathcal{L}_{\mathcal{D}} = \frac{1}{b} \sum_{i=1}^{b} \Big[\log \mathcal{D}\left(X^i_{RGB}, X^i_D\right) + \log\left(1 - \mathcal{D}\left(X^i_{RGB}, \mathcal{G}\left(X^i_{RGB}\right)\right)\right) \Big] \tag{3.1}$$

The loss function (3.1) encourages the discriminator to correctly distinguish between true depth images from the dataset and fake depth images produced by the generator.

Generator

$\mathcal{G}$ will observe a colour image and will produce a related depth image. During training, it will minimize the following loss function:

$$\mathcal{L}_{\mathcal{G}} = \frac{1}{b}\sum_{i=1}^{b}\Big[\log\left(1-\mathcal{D}\left(X^i_{RGB},\mathcal{G}\left(X^i_{RGB}\right)\right)\right) + \lambda_{L1}\cdot L1(\mathcal{G}\left(X^i_{RGB}\right),X^i_D)\Big] \tag{3.2}$$

with λ_{L1} being a custom weight and $L1(X_{D_fake}, X^i_D)$ being the L1 norm between the real depth image and the one generated. Note that the term $\log\left(1-\mathcal{D}\left(X^i_{RGB},\mathcal{G}\left(X^i_{RGB}\right)\right)\right)$ rewards the generator for fooling the discriminator, while the second term $\lambda_{L1}\cdot L1(y_{fake}, y)$ encourages the generator to produce depth images as faithful as possible to the original ones. Also note that the generator will never observe the actual depth images and, therefore, the latter component will push the generator towards the recognition of patterns and identification of distance values directly from the colour image, ideally improving the latent features.

We denote with $\mathcal{N}$ the encoder and with $\mathcal{L}$ its output, i.e., the latent representation. We design the generator as follows:

RGB2Latent: an encoder composed of "down-sampling" blocks, convolutional layers which reduce the input data to a lower size. Each convolutional layer is followed by normalization, max-pooling and dropout layers and it will produce a latent representation, whose size is another hyperparameter.

Latent2Depth: a decoder composed of "up-sampling" blocks. Each block can be composed of a nearest neighbor up-sampling layer, followed by dropout, convolution and normalization layers. We have also experimented with transpose convolution to upscale from the latent space, but we found it to have worse performance in each tested configuration. Therefore, for brevity, we have neglected it in the following.

Note that such a configuration takes inspiration from [52]. The main differences from state-of-art algorithms are in the objective for which such network is trained for (henceforth the type of conversion), which leads to the impossibility of leveraging skip-connections: these type of connections require, at least to some extent, consistency between the input and the output data, which, in this case, is not met.

3.3.2 Point2Latent

The goal of this network is to learn a representation of the input PC in the same latent space as the RGB2Latent encoder. The Point2Latent training process is depicted in Figure 3.3.4. The

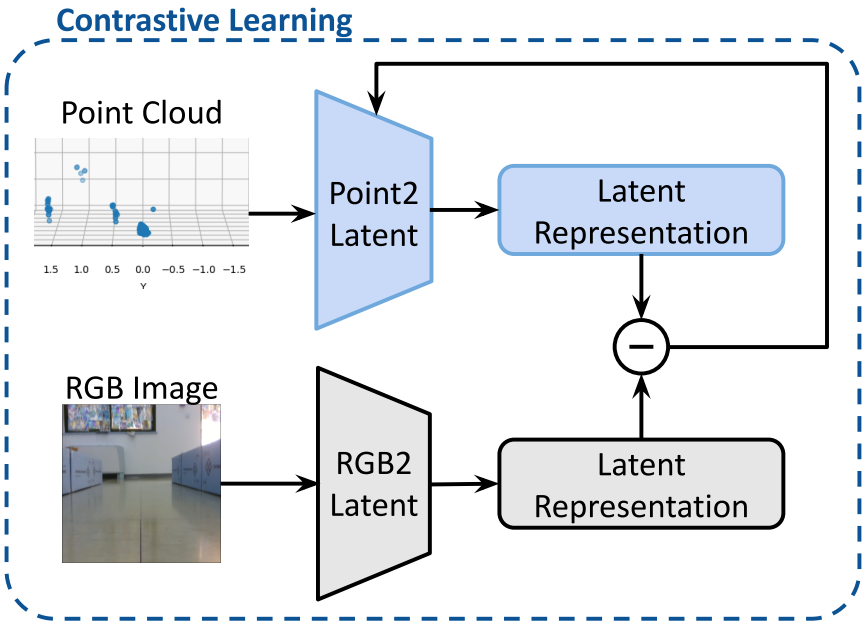


Figure 3.3.4: Point2Latent training process

Point2Latent encoder $\mathcal{O}$ structure design is inspired by the work of [41]. It comprises two T-Nets: the first transforms the input PC into an order-independent representation, while the second extracts useful local features. The T-Nets are interleaved with several convolutional and max-pooling layers. The model's output is produced by a fully connected layer that shares the same output shape of $\mathcal{N}$. An example of this encoder structure is depicted in Figure 3.3.5.

$\mathcal{O}$ is trained following the contrastive learning paradigm. Let X^i_{pc} be the PC associated to the i-th colour (and depth) image. The network will optimize the following loss function:

$$L1(\mathcal{N}\left(X^i_{RGB}\right), \mathcal{O}\left(X^i_{pc}\right)) + ||I - AA^T||^2 \tag{3.3}$$

with I being the identity matrix and A the T-Net transformation matrix.
The term $L1(\mathcal{N}\left(X^i_{RGB}\right), \mathcal{O}\left(X^i_{pc}\right))$ computes the distance between the two representations, while $||I - AA^T||^2$ is a regularization term that leads the T-Net towards an orthogonal transformation. Note that, in order to train this encoder, we first train the RGB2Depth model and then fix its RGB2Latent weights.

3.4 Results

This section presents the results of the proposed approaches, as detailed in Section 3.3.

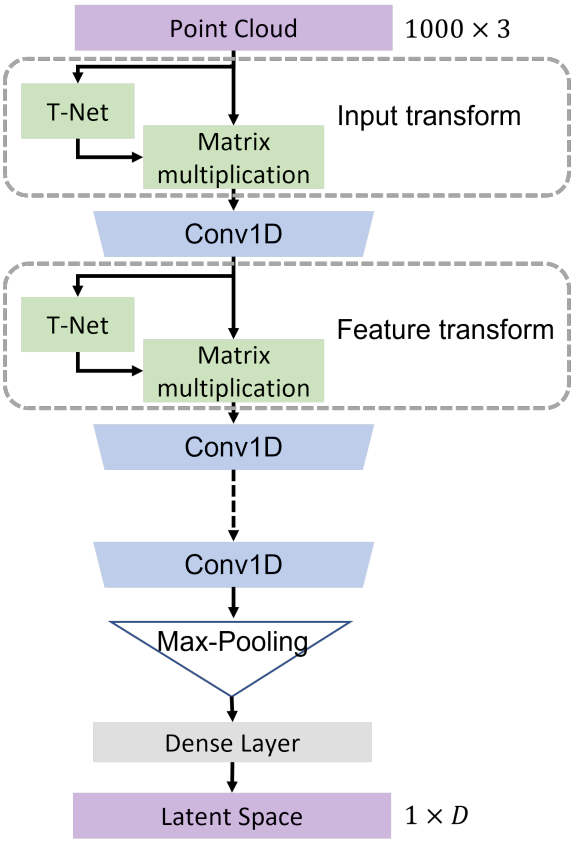


Figure 3.3.5: Point2Latent Structure

3.4.1 Dataset

In order to conduct our experiments and train the networks, we utilized a *Turtlebot3 Waffle*, a differential drive robot that comes equipped with multiple exteroceptive sensors. Specifically, the robot has a RP-LiDAR , a depth camera producing 640×480 frames for both color and depth images, and a mmWave sensor by Texas Instruments. Due to sensor's nature, depth images can present flaws or missing information. The applied pre-elaboration aims at removing missing regions and regularizing data into a $[-1, +1]$ range. Also for point clouds a pre-elaboration is required: we crop points into the camera's field of view; the number of points is fixed to 1000 per frame and the points are normalized, also in this case.

In order to collect relevant and realistic data, the robot was tasked with navigating a 6×6m area filled with obstacles, capturing poses of both the robot and the obstacles via the Vicon Tracker[1]. A variety of obstacles were utilized, including *standard* ($0.49 \times 0.36 \times 0.25$m), *medium* ($0.36 \times 1.47 \times 0.25$m), *long* ($1.96 \times 0.36 \times 0.25$m), and *high* ($0.23 \times 0.95 \times 0.45$m) obstacles.

3.4.2 Performance Analysis

In order to study and validate the performance of the proposed approach, we considered several network configurations. Note that the performance of the GAN is not influenced by the Point2Latent encoder, but the latent representation identified by the auto-encoder strongly impacts the Point2Latent training results. In fact, the RGB2Depth auto-encoder may learn a representation that is not suitable for the Point2Latent encoder. Therefore, when a good RGB2Depth configuration is found, several tests are conducted on the Point2Latent parameters. Many preliminary tests are conducted to reduce the number of hyperparameters left to identify. In the remaining, we will use "Adam" optimizer with 0.0001 as learning rate, a λ_{L1} equal to 100 and each model is trained for 200 epochs. In the following section, we introduce the results of the tests conducted on the cGAN model and evaluate the Point2Latent training performance.

RGB2Depth

The aim of this model is to identify a significant latent representation which will be later shared with the Point2Latent model. For the sake of brevity, only the most significant tests are reported, although many other hyperparameters have been tuned and altered, and several configurations resulted in ineffective models.[2]

[1] https://www.vicon.com/software/tracker

[2] To simplify the notation, down and up -sampling blocks are named with the initials of each type of layer of which they are composed. A prefix number will specify the number of such blocks, while an afterword number will specify the size of the latent space (e.g. an auto-encoder with 8 down-blocks and 8 up-blocks will be referred to as

The best-performing models are achieved with a latent space with 512 features, using both 8 and 4 down-sampling blocks and 8 up-sampling blocks , as shown in Figure 3.4.1(a). Although their learning curves almost overlap, we observed that the two networks produce distinct high-level performance. A lower number of down-sampling blocks results in an ineffective extraction of useful information, leading to a decoded depth image with several artifacts and inaccuracies. For these reasons, we consider the model with 8 down-sampling blocks as the backbone model for the RGB2Depth in the following tests.

Point2Latent

As previously mentioned, this network is inspired by [41], particularly its classification branch, but with a modified last layer to suit the purpose.Based on the results of training the RGB2Depth model, we fix the latent representation to 512 features. In order to deterimine the optimal configuration, we repeat the training while changing the activation function (linear: l, ReLU: R, Tanh: T, Sigmoid: S) and applying (A) or not (N) normalization to the latent layer. These two letters are postponed to the notation used for experiments. The results are presented in Figure 3.4.1(c).

The best performing configurations employ all four types of activation functions on the latent layer: linear, sigmoid, tanh, and ReLU, with the first two models using normalization, namely lA-A, SA-A, TN-N, and RN-N. In order to evaluate the generalization capabilities, we test these models on a separate test set and compute the mean and standard deviation (Std.D.) of two error metrics: L1 norm and Mean Squared Error (MSE). As shown in Table 3.1, all models perform similarly, including those that performed poorly in the training phases. For this reason, it is necessary to conduct a practical study of the model's performance. During preliminary tests, we observed that several models, even those with good performance, produced depth images with inaccuracies and artifacts that were not captured by the considered metrics. Further investigation revealed that the best performing model, which produced accurate depth images without artifacts, is the RN-N model. An analysis of results shown in Figure 3.4.2 reveals two main findings: (1) model RN-N is capable of representing the environment with a good accuracy, with the obstacle on the left and an open area on the right, while the remaining models fail to detect the obstacle altogether; (2) even in only slightly adverse environmental conditions, the real depth image can be very misleading, emphasizing the importance of using a sensor that is robust to such conditions for optimal perception of the environment and safe navigation. In order to further study the generalization capabilities and effectiveness of the identified best performing models, additional tests were conducted and are discussed in the following.

"8CND-512-8UCND", where "512" identifies the latent size.

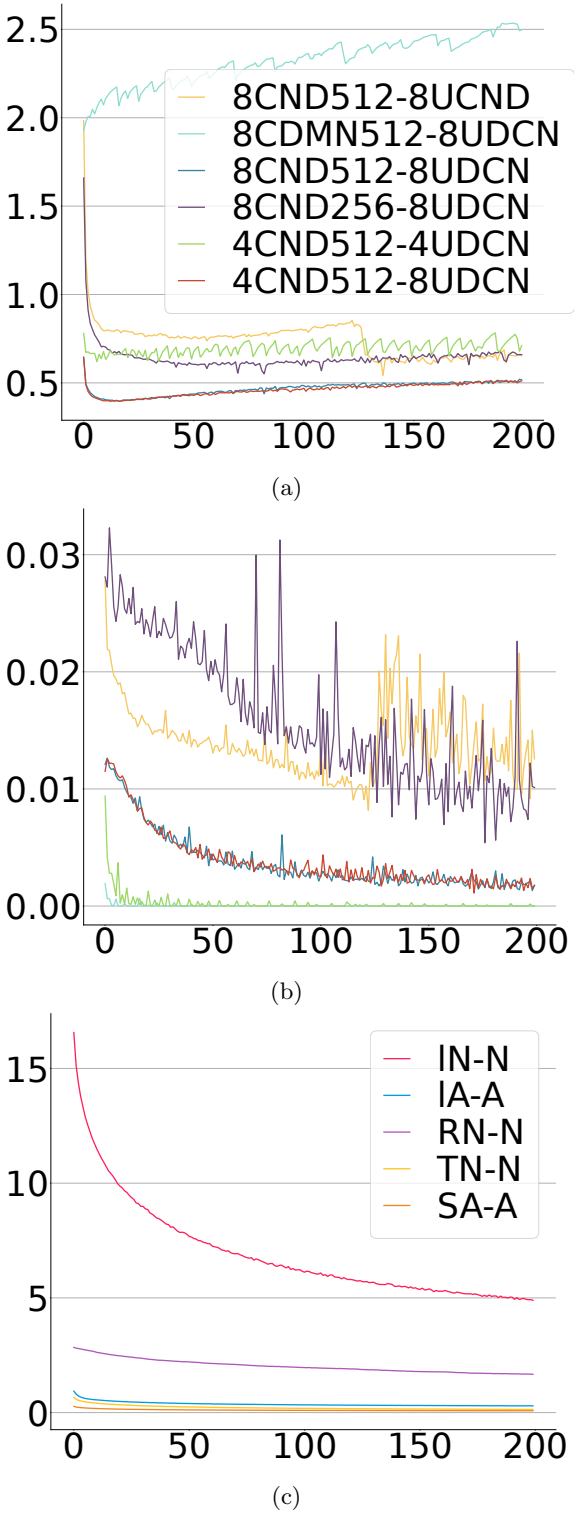
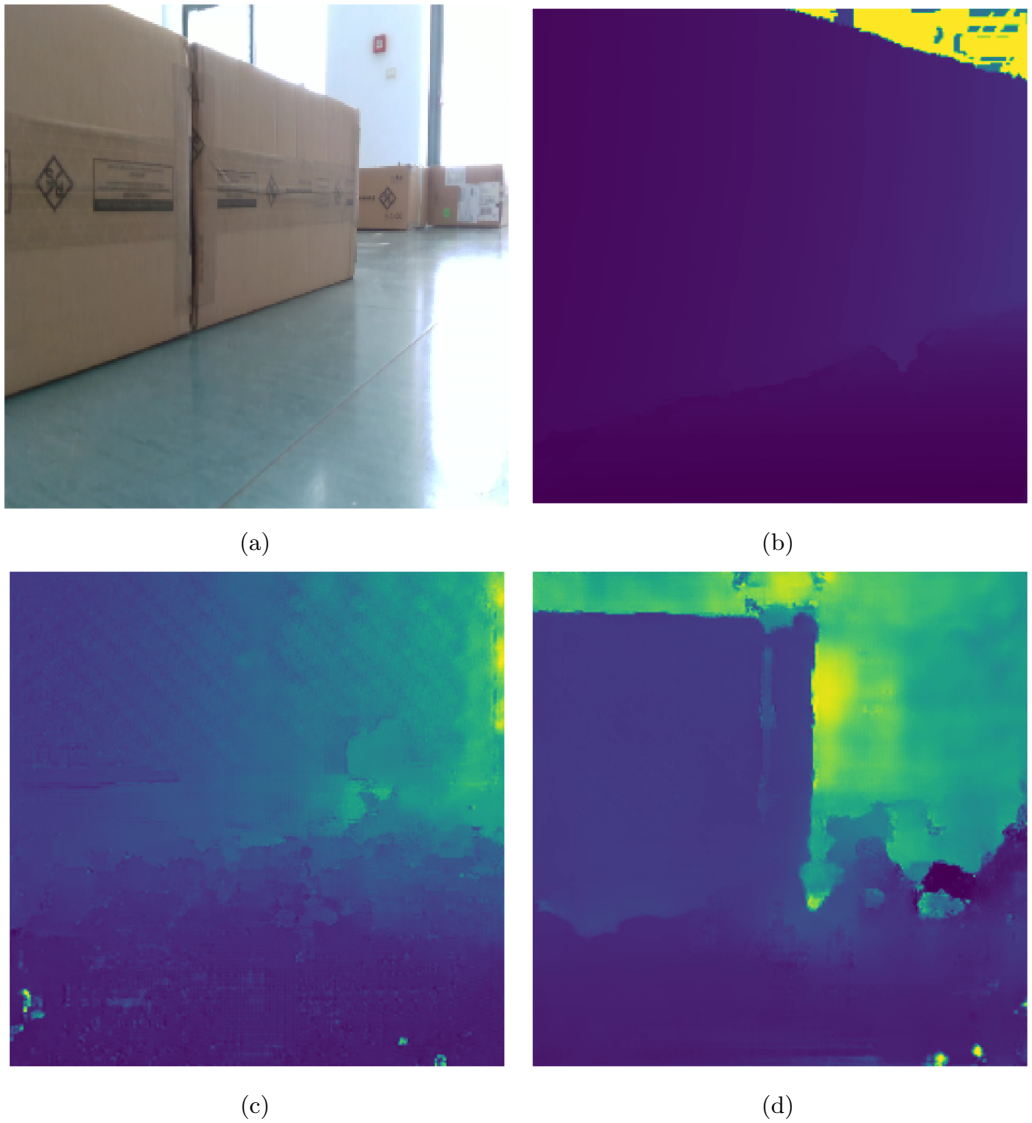


Figure 3.4.1: Training Losses for Generator (a), Discriminator (b) and Point2Latent (c)

Table 3.1: Point2Depth test errors

Model	L1	L1 Std.D.	MSE	MSE Std.D.
lN-N	639	374	1114061	968965
lA-A	635	370	1108748	938931
RN-N	660	405	1191768	1039095
TN-N	658	380	1140771	935083
SA-A	650	377	1161711	955506



(a) (b)

(c) (d)

Figure 3.4.2: Real Colour Image (a), Real Depth Image (b), SA-A prediction (c) and RN-N prediction (d)

3.4.3 Experiments

In this section, we investigate the performance and limitations of the best-performing model by considering two insightful scenarios. These tests aim to provide practical insights into the model's performance. For each test, we provide three images: the RGB image captured by the camera, the related depth image, and the point cloud captured by the mmWave sensor. Additionally, two figures report the predictions of both the RGB2Depth and Point2Depth networks of the RN-N model for each scenario.

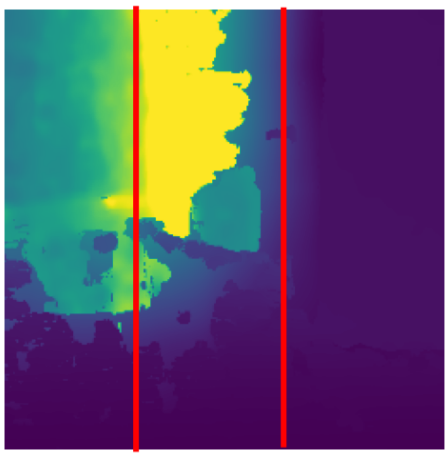
Test 1

in this test, the robot is facing three different obstacles. The first obstacle (right of second red line) is a known obstacle observed during training. The second obstacle (left of the first red lines) is a row of desks covered with white thin paper, which is particularly challenging since it is not easily detectable by the mmWave sensor. The third obstacle (between the two red lines) is an unseen object, another robot, much more complex than objects seen during training (see Figure 3.4.3). Figure 3.4.4 shows the RGB2Depth and Point2Depth predictions. The RGB2Depth prediction appears similar to the original depth image shown in Figure 3.4.3(b), yet a key piece is missing: the unseen object cannot be associated with any of the regions between the two red lines. On the other hand, the Point2Depth prediction introduces distance values associated with the distance from the object, even if it resembles a box (an object seen during training). This is an interesting finding that we ascribe to two key reasons: (1) the RGB2Depth is actually trained to find the most effective latent representation, rather than a faithful depth image; (2) the mmWave PC contains information of such object and the Point2Latent model is able of conveying such information effectively into the latent space, allowing the Latent2Depth model to represent such information into the decoded image. However, the Latent2Depth decoder is not able of properly representing such object, as it has never seen it during training, replacing it with a familiar obstacle. Even if not optimal, we consider this result still useful, especially for navigation purposes.

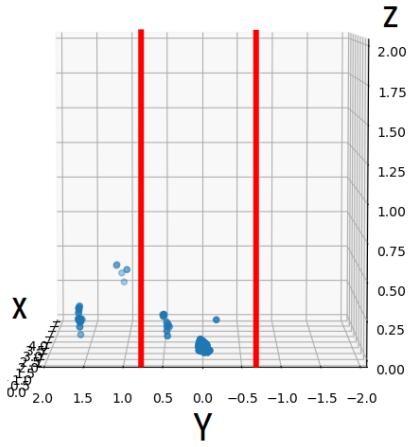
Test 2

the robot is facing directly the row of desks covered with paper. Figure 3.4.6 shows both the RGB2Depth prediction and the Point2Depth one. The RGB2Depth prediction is faithful to the original depth image shown in Figure 3.4.5(b). On the other hand, the Point2Depth prediction is far from the original image, appearing as an image with an obstacle on the left and free space on the right. This result is expected as the model is observing an uninformative PC as shown in Figure 3.4.5(c)). In fact, the white paper is not detected and the points in the cloud are coming

(a) Colour Image

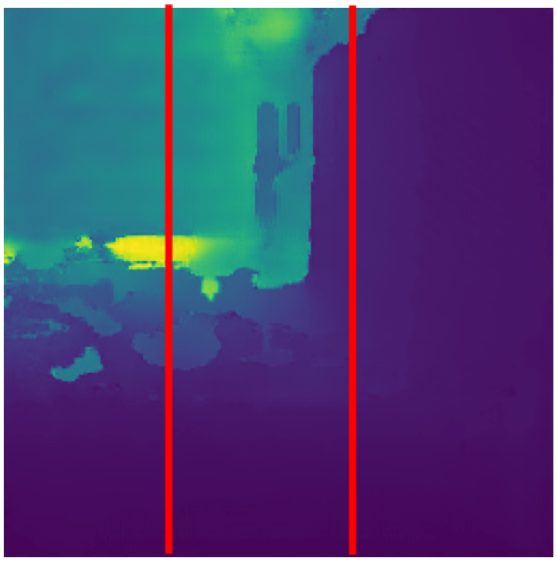


(b) Depth Image

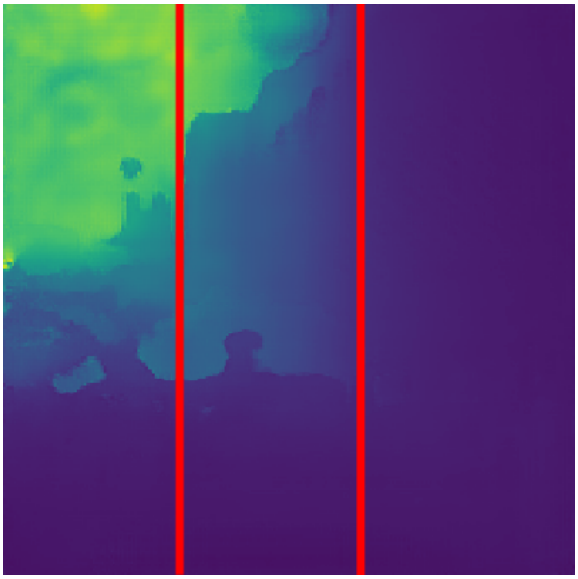


(c) Point Cloud

Figure 3.4.3: Test 1



(a) RGB2Depth



(b) Point2Depth

Figure 3.4.4: Test 1 - Model predictions

from the desk behind the paper. Therefore, while the prediction is far from the original depth image, it is actually producing a faithful image with respect to the input PC. Of course, the result is still not acceptable, but the goal of such a model is not to make up information that the mmWave sensor is not producing.

3.5 Quantizing Point2Depth

This section presents the preliminary results of a collaboration with STMicroelectronics. The idea is to bring the trained Point2Depth model to an embedded device.

We start by providing some information regarding the trained model and then we move to the target devices.

Memory footprint: the trained model, whose performance have been examined in Section 3.4, has a footprint memory of 54.88MB. While it may not seem a big model, one has to keep in mind that embedded devices, such as the STM32F446RE[3], usually have few hundred kilobytes of *flash memory*. For instance, the STM32F446RE board only has 512KB of flash memory. With such limited memory, loading the trained model becomes impossible.

Inference Time: on the laptop used to train the model equipped with 32GB of RAM, an Intel i9-13900HX with 24 cores and an NVIDIA RTX4070 with 16GB of VRAM, the inference time lasts, on average, 0.043 seconds. While the inference time is very low, such a laptop is far from being an embedded device.

Target Devices

In order to bring the study conducted so far closer to embedded scenarios, we select four embedded boards which are usually employed in industrial and robotics applications.

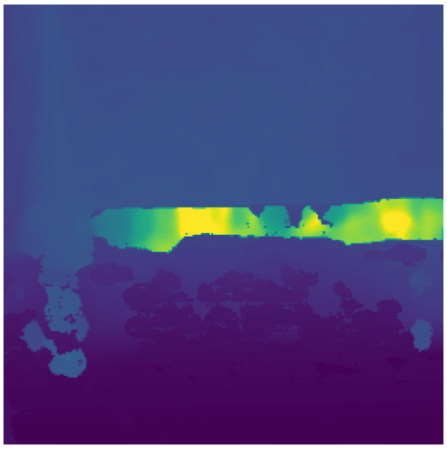
The *NVIDIA Jetson Xavier NX*[4], equipped with 16GB of RAM, a 6-core NVIDIA Carmel Arm v8.2 64-bit CPU and a GPU NVIDIA Volta architecture with 384 CUDA cores, reaching up to 21 Tera Operations Per Second (TOPS). Despite its high-end characteristics, its small size (70mm $\times$ 45mm) made it very popular in robotics applications.

Then, we select three boards from STM32 microprocessors working at three different levels, which are typically employed in many contexts from industrial automation, to smart houses and medical healt-care.

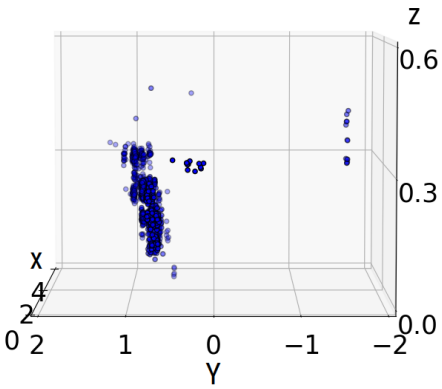
The low-end *STM32MP135F-DK*[5]: STM32MP135 microprocessors (MPUs) are based on a single

[3] https://www.st.com/en/microcontrollers-microprocessors/stm32f446re.html
[4] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/
[5] https://www.st.com/en/evaluation-tools/stm32mp135f-dk.html

(a) Colour Image

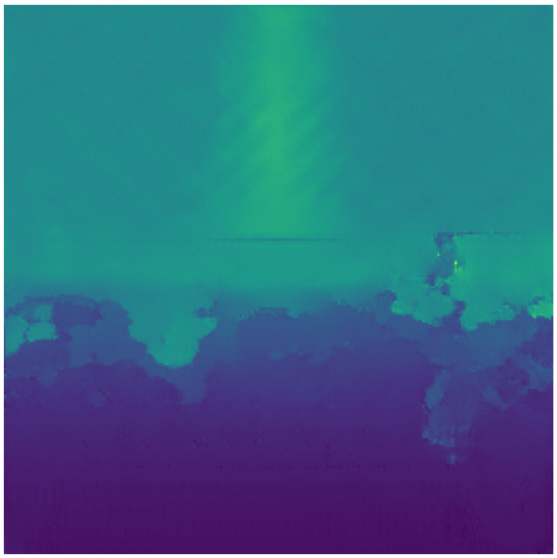


(b) Depth Image

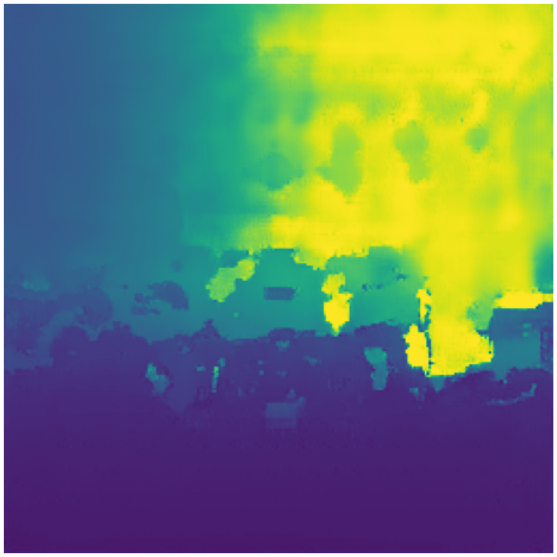


(c) Point Cloud

Figure 3.4.5: Test 2



(a) RGB2Depth



(b) Point2Depth

Figure 3.4.6: Test 2 - Model predictions

Arm Cortex-A7 core running up to 1GHz and have a 4Gbit DDR3L memory module.

The mid-end *STM32MP157F-DK2*[6]: this MPU is from the same family of the STM32MP135F-DK. However, it leverages a two core CPU, which enables parallel processing on an embedded devices.

The high-end *STM32MP257F*[7]: this board leverages a Arm Cortex-A35 and Arm Cortex-M33 working at 1500 MHz and a 16-Gbit DRAM. This board comes with an increased memory, and its two cores makes it for a very powerful embedded device.

The chosen platforms all have enough memory to hold the model. However, the CPU performance are far from the ones of the laptop on which the training was carried out. It is worth evaluating the inference times of the Point2Depth on such boards. The results are presented in Table 3.2. Clearly, such performance are not enough, especially in real time constrained cases such as the one of controlling a robot.

For this reason, *quantization* of neural networks has been taken into account as a possible solution to the task.

As introduced in Section 2.3, quantization is a technique that brings neural networks from the original bit representation (typically 32 bits), into a lower bit representation.

In the literature there exists extreme approaches that bring neural networks into a 2 or a 3 bits representations ([56, 57]), that dramatically reduce both the memory footprint and the inference time of the original neural network. However, these approaches are also associated with much lower performance. Another important consideration is that, to effectively leverage such low representations, the device should be equipped with a processing unit capable of performing arithmetical operations directly in that representation. In general, if the CPU is not equipped with such lower bit processing unit, it can still be possible to carry out the operations by casting the values into a higher bit representation, compute the result and cast it back to the lower representation. However, such solution may come with approximation errors and, in general, it is slower than CPU-native operations. Given that three out of four embedded devices are not equipped with such low bit architectures, a quantization to an 8 bit representation is chosen. Furthermore, the Quantization Aware Training has been discarded in order to try to exploit the model obtained in the previous sections.

[6] `https://www.st.com/en/evaluation-tools/stm32mp157f-dk2.html`
[7] `https://www.st.com/en/evaluation-tools/STM32MP257F-EV1.html`

Table 3.2: Point2Depth vs Quantized Point2Depth Inference Times

	Point2Depth Inference Time [s]	Quantized Point2Depth Inference Time [s]
Laptop	0.043	0.0287
Jetson Xavier NX	1.22	1.035
STM32MP135F-DK	14.30	6.097
STM32MP157F-DK2	8.328	4.024
STM32MP257F	2.878	1.345

PyTorch Post Training Quantization

To quantize the model, PyTorch[8] APIs have been used. In fact, PyTorch offers support for INT8 quantization, which enables a 4x reduction in model size compared to standard FP32 models.

To effectively carry out the quantization, the model has been wrapped by a *QuantStub* and *DeQuantStub* modules, which enable proper quantization of, respectively, the input and output data. Then, the pairs convolutional-batch normalization and fully connected-batch normalization layers have been *fused*. The resulting model undergoes a *preparation* phase, that carries out the calibration of all the quantization data.

Then, the model is converted following PyTorch APIs[9]. The resulting model, hereafter named QPoint2Depth, is now effectively quantized into an 8 bit representation. Its memory footprint is reduced just 14.6MB, which is, as expected, roughly one fourth of the original size.

The resulting model has been tested on the target devices as well as on the laptop used to train the model. In particular, for each device, the inference time has been registered and the results are reported in Table 3.2.

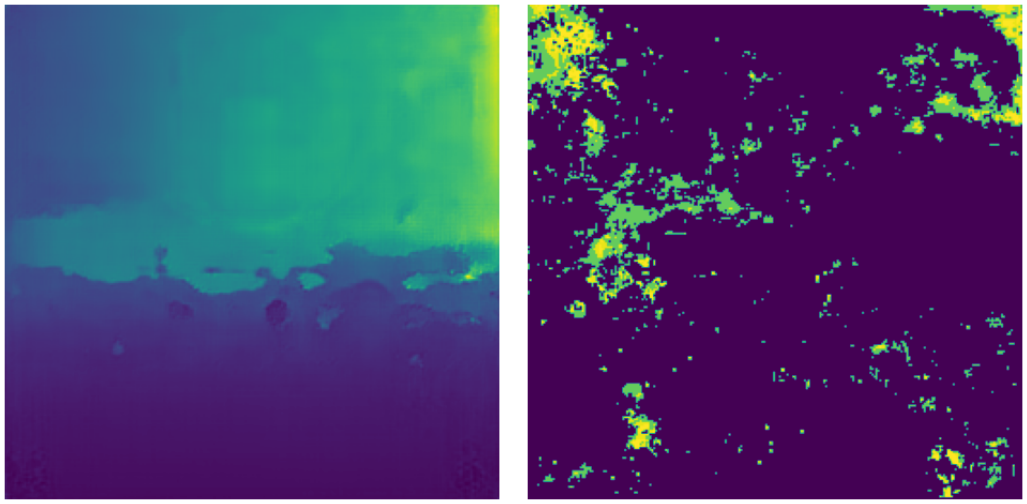
As expected, the quantization improves the inference time, halving it on every platform. The STM board taht lands itself to better performance is, clearly, the STM32MP257F, with only 1.345 seconds of inference time. Considering that these are extremely low end boards, this is an encouraging result. In fact, leveraging this device one could reach roughly 0.75 conversion per second. A different trend is observed with the NVIDIA board, were the quantized model only gains few hundreds milliseconds with respect to the original model. Nonetheless, QPoint2Depth still reaches roughly 10 conversions per second, which allows for real time applications of the quantized model.

Experiments

In order to validate the quantized model, a qualitative analysis is conducted. In particular several frames have been compared between Point2Depth and its quantized version.

[8] `https://pytorch.org/`

[9] `https://pytorch.org/docs/stable/generated/torch.ao.quantization.convert.html#torch.ao.quantization.convert`



(a) Point2Depth (b) QPoint2Depth

Figure 3.5.1: Point2Depth vs QPoint2Depth - frame 3

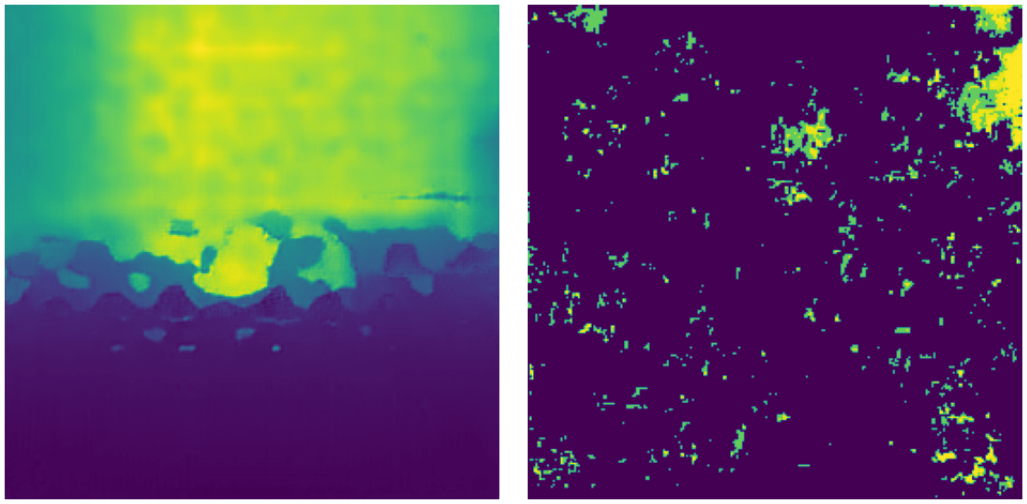
Figures 3.5.1 and Figure 3.5.2 report two frames (frame 3 and frame 367) from a test run. Despite QPoint2Depth runs twice as fast Point2Depth and has a fourth of its memory footprint, its predictions has far from acceptable. The depth images obtained from QPoint2Depth are useless, since it is not possible to identify any obstacles, or objects.

This result can be connected to two main reasons: 1) the complex architecture of Point2Depth takes advantage of more than just the 8 bits used by QPoint2Depth; 2) applying post training quantization is a trivial approach that does not take into account of any architecture peculiarities.

3.6 Concluding Remarks

Perception of the surroundings is a crucial task for mobile robotics, enabling robots to navigate their environment safely while fulfilling high-level tasks and avoiding obstacles. LiDARs, RGB and RGB-D cameras are popular sensors for this purpose due to their precision and ease-of-use. However, working in the visible light, they can be impaired to the point of being non-usable when utilized in harsh environments. In contrast, mmWave sensors are less affected by environmental factors due to their use of physical principles. As a result, they are gaining increased attention from both industry and academia. However, mmWave sensors produce sparse Point Clouds (PCs).

In this chapter, a cross-modal contrastive learning approach based on cGANs is proposed to translate mmWave PCs into depth images. A multimodal dataset containing strongly correlated color images, depth images, and mmWave PCs has been built. Several tests have been carried



(a) Point2Depth (b) QPoint2Depth

Figure 3.5.2: Point2Depth vs QPoint2Depth - frame 367

out to determine the best topology and hyperparameter configuration for each neural network involved in the approach. Further tests were conducted to evaluate the effectiveness of the best performing model. The results demonstrate that the final proposed model can effectively produce information similar to depth images and can generalize to unseen objects, even when actual depth images cannot provide the necessary information. After obtaining a good model, quantization is studied as an approach to bring this model on embedded devices. The results show that quantization is a good resource when it comes to reducing the memory footprint and inference time of neural networks. However, practical experiments reveal that blindly quantizing the model after training can lead to suboptimal performance from a qualitative point of view. As a further investigation, quantization aware training can be carried out in order to obtain a model that fully leverages the 8 bit representation, learning how to deal with such limitation while producing faithful depth images. Another interesting research direction can be analysing the performance of the proposed model in dynamic and more complex environments, improving its robustness to unseen objects.

Chapter 4

MilliNoise: a Millimeter-wave Radar Sparse Point Cloud Dataset in Indoor Scenarios

4.1 Introduction

In the evolving landscape of multimedia systems, the role of 3D data sources has grown significantly, requiring new ways to capture, process, and utilize information. *PC* data, produced by 3D LiDAR sensors and stereocameras, is a powerful instrument for capturing the features of three-dimensional scenes. These technologies have induced significant advancements in applications encompassing autonomous navigation and robotics [58,59], 3D modelling [60], augmented reality/virtual reality [61–63], and 6DoF video streaming [64, 65].

Despite their advantages, LiDAR and stereocamera systems are often challenged by environmental conditions, such as poor visibility in adverse weather or difficulties in detecting transparent or reflective surfaces [42, 66]. This limits the utilization of such media content for reliable machine applications. Furthermore, these technologies may be cost prohibitive and pose concerns regarding power consumption, thus hindering their adoption in mobile applications.

MmWave radars are sensors that operate at wavelengths of the order of millimeters. They are equipped with transmitter antennas that emit signals and receiver antennas that capture reflections of these signals. The processing of received signals enables the generation of a 3D PC that describes the environment, including information on the velocity, intensity and orientation of the reflecting objects. Operating within the high-frequency spectrum, typically ranging from 30 to 300 GHz, mmWave radars are capable of detecting multiple objects, even when they are obstructed by other

elements. Moreover, these sensors are more robust in adverse environmental conditions, such as fog, dust, smoke, and rain [58, 62, 66, 67]. In this paper, we consider *FMCW* mmWave sensors with integrated antennas, referred to as *Antenna-On-Package* (AoP). Unlike conventional pulse radar sensors, FMCW sensors periodically and continuously emit chirp signals, instead of short pulses [68]. AoP sensors allow a compact package, resulting in significantly reduced production and retail costs, which helped increasing the adoption of mmWave sensors in several research and industrial fields.

Even though mmWave sensors can be considered as an interesting alternative and complement to LiDAR and stereocamera systems, especially in mobile scenarios, the PC data they generate introduces several peculiar challenges. Noise in PC data, usually caused by sensor limitations, environmental factors, or inherent system noise, can significantly degrade application quality. Moreover, unlike 3D LiDAR and stereocamera, these sensors provide sparser PCs, with a reduced amount of details for each detected object. In short, mmWave data suffers from key limitations, specifically (i) the presence of noise and (ii) the sparseness of the acquired data, which substantially limit the adoption of these data sources for multimedia systems. It is therefore an open question if we can use such PC data for machine processing.

Contributions: This chapter collects accurately labelled sparse PCs generated by FMCW mmWave radar sensors, to allow the design of learning-based denoising approaches. To this end, MilliNoise is presented, a dataset that collects different indoor scenarios, such as wide and narrow hallways, tight and loose turns, and shelves, built by properly arranging a set of obstacles. Each point in the MilliNoise dataset is accurately labelled through a motion tracking system, allowing the discrimination of points describing actual objects in the scene from those representing noise, with a sub-millimeter accuracy. The dataset also includes the intensity and velocity values for each point captured by the mmWave sensor. Furthermore, point-wise distances to the closest obstacles are provided, enabling the application of regression methodologies for denoising the MilliNoise dataset.

4.2 Background on mmWave radars

In this section, we briefly present the working principles of mmWave FMCW radars, such as the one used to collect the MilliNoise dataset, highlighting the way PCs are obtained by processing the radar's raw data and the main sources of noise. The interested reader is referred to [69] for a comprehensive treatment of mmWave FMCW radar processing.

Working Principles: In a nutshell, the sensor emits a set of signals to obtain information about the environment. Each signal, referred to as *chirp*, is a sinusoidal wave with a frequency that increases linearly over time. A set of such signals then composes the *chirp frame*. Each receiver builds a

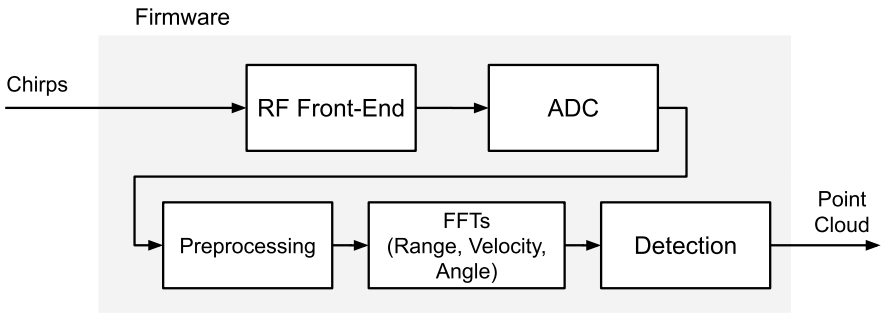


Figure 4.2.1: Sensor Pipeline employed to produce PCs from sensor's readings

two-dimensional matrix of N_c chirps per frame by N_s samples per signal. Then, by joining all these matrices, a three-dimensional matrix, denoted as a *radar cube*, is built.

The pipeline employed to produce the PCs from the emitted chirp frames is shown in Figure 4.2.1. The radar cube allows to extract information such as: the distance and angle of points belonging to objects in the scene from which the (x, y, z) coordinates can be computed, the points' intensities and velocities.

In particular, the radar cube is analyzed using the Fast Fourier Transform (FFT). The distance of a given obstacle is associated with each peak in the cube: the further away the sensor is, the longer the delay between the emitting and receiving peak [69]. Instead, the value of each peak identifies the intensity of the related point.

The angle estimation is obtained by considering the angle of arrival of the same emitted signal on multiple antennas. By leveraging the known physical displacement between antennas, angle can be extracted by measuring the phase shift of the identical signal received by different antennas. The radar cube allows estimating velocity information of the objects by utilizing multiple chirps. In particular, the velocity of the object reflecting the emitted waves induces a phase shift of the signals from the same chirp on the same antenna. Unlike angle estimation, which involves multiple receivers, this approach leverages the measurements of the phase shift from a single antenna. Further processing of this phase shift through the Doppler-FFT, which accounts for the Doppler effect in the frequency domain, allows the estimation of the velocity of the observed objects. The final PC will contain a number of points depending on the number of reflections captured by the receiving antenna. As a consequence, this procedure generates PCs with a time-varying number of points.

Main challenges: Similarly to any other PC data, mmWave PCs are a set of unordered points that lack correlation with their position in the data structure, thereby inhibiting the application of processing techniques reliant on this type of positional information.

However, unlike other PCs, mmWave ones present peculiar limitations that pose new challenges

for data processing. When dealing with low-cost mmWave FMCW AoP radars, constraints on antenna size and their maximum displacements impose limitations on angle and range accuracy. This limitation is evident in the resulting PCs, where the detected points exhibit improved accuracy when the angle is sufficiently small (typically in the $[-10°, 10°]$ range in practical scenarios) [70]. Moreover, mmWave sensors are susceptible to noise, presenting randomly scattered points or artificial reflections, such as clutter and multipath reflections. This challenge becomes more prominent in confined indoor spaces where *ghost* objects and multipath rays may appear. Effectively addressing noise is crucial to improve the signal-to-noise ratio and mitigate the risk of misinterpreting these artifacts as false positives in the processing pipeline. Therefore, the importance of the MilliNoise dataset, which focuses on indoor scenes, becomes evident. Furthermore, mmWave PCs are highly sparse, reaching a few hundred points per frame, instead of the typical several thousand points produced by depth-cameras and LiDARs [66].

4.3 Related Work

Table 4.1: Comparison of mmWave Datasets for autonomous navigation tasks

Dataset	**Runs**	**Frames (Points)**	**Environment**	**Annotation**
nuScenes [71]	1000	1.3M (N/A)	Outdoor	Bounding Box
Astyx [72]	N/A	500 (N/A)	Outdoor	Object oriented
CARRADA [73]	N/A	12.6k (35M)	Outdoor	Bounding Box
RadarScenes [74]	158	N/A (N/A)	Outdoor	Point-Wise
Ghost Dataset [75]	111	N/A (35M)	Outdoor	Point-Wise (manual)
OdomBeyondVision [76]	119	N/A (N/A)	Indoor	Not annotated
MilliNoise (ours)	49	58k (11.6M)	Indoor	Point-Wise sub-mm accuracy

Existing mmWave radar datasets can be grouped into two main classes: datasets for human sensing and datasets for autonomous navigation. In the former, the datasets [63, 77–80] consist of sequences of human activities acquired using a static mmWave radar. They primarily serve as a base for action recognition applications. By contrast, autonomous navigation datasets, including ours, involve a mmWave radar mounted on a moving system, such as a robot or vehicle, to capture the surrounding 3D space as it moves. The acquired data are then processed for autonomous navigation tasks. These navigation datasets are often multimodal, with mmWave sensors combined with LiDAR or RGB/thermal cameras for richer information. In this section, we provide an overview of PCs datasets acquired through mmWave sensors for autonomous navigation systems. A summary of the main features of datasets containing mmWave PCs for autonomous navigation scenarios is presented in Table 4.1 which also includes the MilliNoise dataset presented in this paper.

Over the last few years, research on autonomous navigation has benefited from several large-scale

LiDAR datasets, such as *KITTI* [81] and *Waymo* [82]. These datasets capture outdoor automobile scenes as dense PCs and come with high-fidelity annotation tools [83]. By comparison, radar datasets are not as high-quality due to the limitations of FMCW radars. More specifically, they suffer from significantly more noise and are difficult to provide ground-truth annotations. The *nuScenes* [71] is a multimodal dataset also containing radar data. It provides 1000 outdoor scenes, each lasting 20s, with objects annotated with bounding boxes. *Astyx* [72] is a smaller dataset of 500 frames containing around 3000 labeled 3D objects. Data are semi-automatically labeled and manually refined to provide ground truth annotations. In *CARRADA* [73], the dataset is collected using a mmWave sensor synchronized with an RGB camera. The dataset provides a total of 12666 frames, 7193 of which are annotated. Here, annotations are made by a deep-learning method and tracked by the SORT algorithm [84]. Similarly, in [85], an automatic label generation based on position, velocities, and previous semantic annotation for *RadarScenes* [74] dataset is proposed.

Given the difficulty in providing annotations for mmWave data in the above datasets, the annotation process is often simplified. The common approach is either to use a second modality as a reference (e.g. LiDAR) or based on acquisition characteristics (e.g. based on the Doppler effect [86]). However, these types of labeling strategies may lack precision and are therefore not guaranteed to be accurate. Within this context, the *Ghost-Data* [59] dataset offers mmWave PCs manually annotated. The *Ghost-Data* dataset is an outdoor dataset of 21 automobile-rehearsed scenarios of the main object (a pedestrian or cyclist) moving near one or two reflective surfaces. In-depth knowledge of the scenes allows manually labeling of the points. The points are not only labeled as real or noise but the noise points are also labeled according to first or second-order reflection. However, not all points were labeled. A significant number of points were simply labeled as "background".

It is important to note that all the navigation datasets presented so far have been acquired from outdoor scenes. In the current literature, there is a gap in datasets of indoor environments for autonomous navigation. One of the few datasets available for indoor scenes is the OdomBeyondVision [76]. However, this dataset was designed specifically for the ego-motion estimation task. The OdomBeyondVision points are not labeled, which limits research for navigation tasks beyond ego-motion estimation, such as denoising or obstacle detection.

The MilliNoise dataset seeks to fill this gap in the literature by providing point-wise labelled scenes recorded in an indoor environment. MilliNoise is the first mmWave PC dataset of *labeled indoor* scenes on long trajectories and with multiple obstacle positions. Furthermore, given our knowledge of the environment, the labels of the MilliNoise are accurate with sub-millimeter precision and guaranteed to be correct.

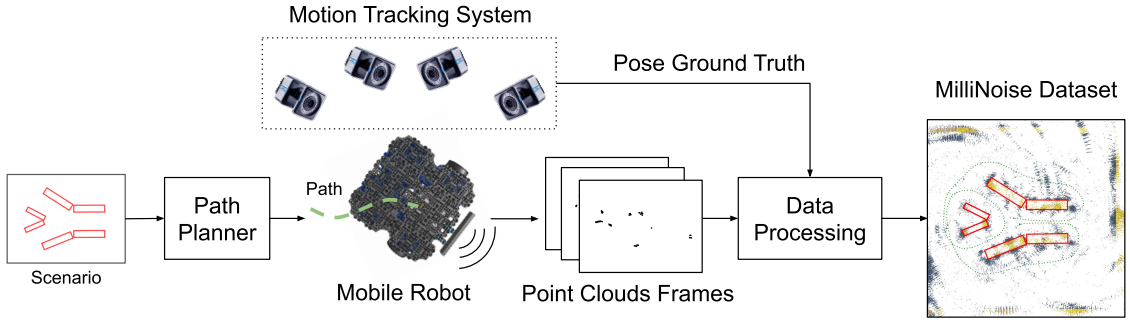


Figure 4.4.1: The MilliNoise dataset acquisition system

4.4 The MilliNoise dataset

This section describes the acquisition and pre-processing steps used to create the MilliNoise dataset, followed by a description of the dataset features, organization and tools to access and manipulate it.

4.4.1 Capturing MilliNoise

The MilliNoise dataset has been captured using the acquisition system shown in Figure 4.4.1. The data has been collected via a mmWave radar sensor mounted on a mobile robot which followed several paths generated by a path planning algorithm in indoor scenarios populated by obstacles placed in an arena equipped with a motion tracking system.

mmWave Sensor Setup: To capture the surrounding 3D space, a differential drive robot, the *Turtlebot3 Waffle Pi*[1], has been equipped with the Texas Instruments' mmWave sensor AWR6843AOP[2]. As shown in Figure 4.4.1, the sensor has been mounted in the front of the robot. The collected data includes not only the points' coordinates but also intensity and velocity values for each point (see Section 4.2).

Motion Tracking Setup: Besides the mmWave sensor data, the positions of both obstacles and robot are recorded using *Vicon Tracker*[3] motion capture system. This system, based on an array of 8 infrared cameras, utilizes passive reflective markers to detect and recognize objects in the scene, providing objects' pose measurements with an error below a tenth of a millimeter.

Scenario Setup: The idea is to emulate realistic indoor environments, such as wide and narrow hallways, tight and loose turns, shelves and so on. To reach this goal, six courses populated by several obstacles were implemented using boxes of different sizes in a 6×6 meters arena, thus generating various possible *scenarios*.

[1] https://www.robotis.us/turtlebot-3-waffle-pi/
[2] https://www.ti.com/product/AWR6843AOP
[3] https://www.vicon.com/software/tracker/

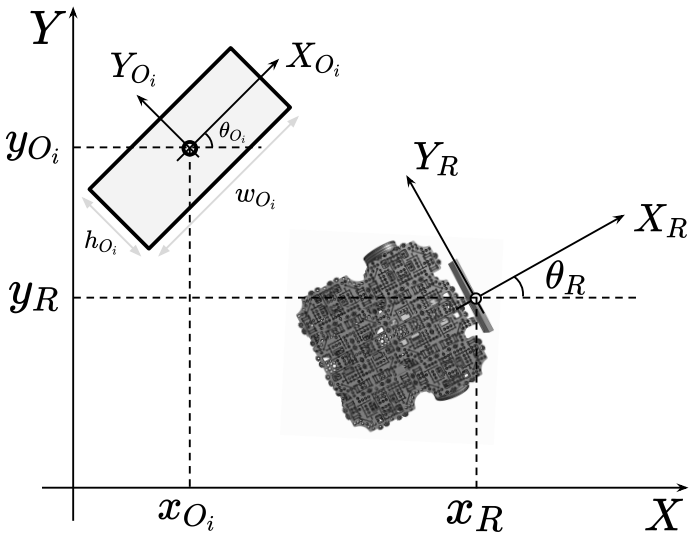


Figure 4.4.2: Scenario definition

Formally, a scenario is defined by a set of obstacles O_i $(i = 1, \ldots, N)$ properly arranged in the environment. Figure 4.4.2 depicts the robot next to the i-th obstacle and reports the notation used to define a scenario. XY is the coordinate system of the motion tracker, which is defined as the *global coordinate system*. Each obstacle O_i in the scene is defined as follows:

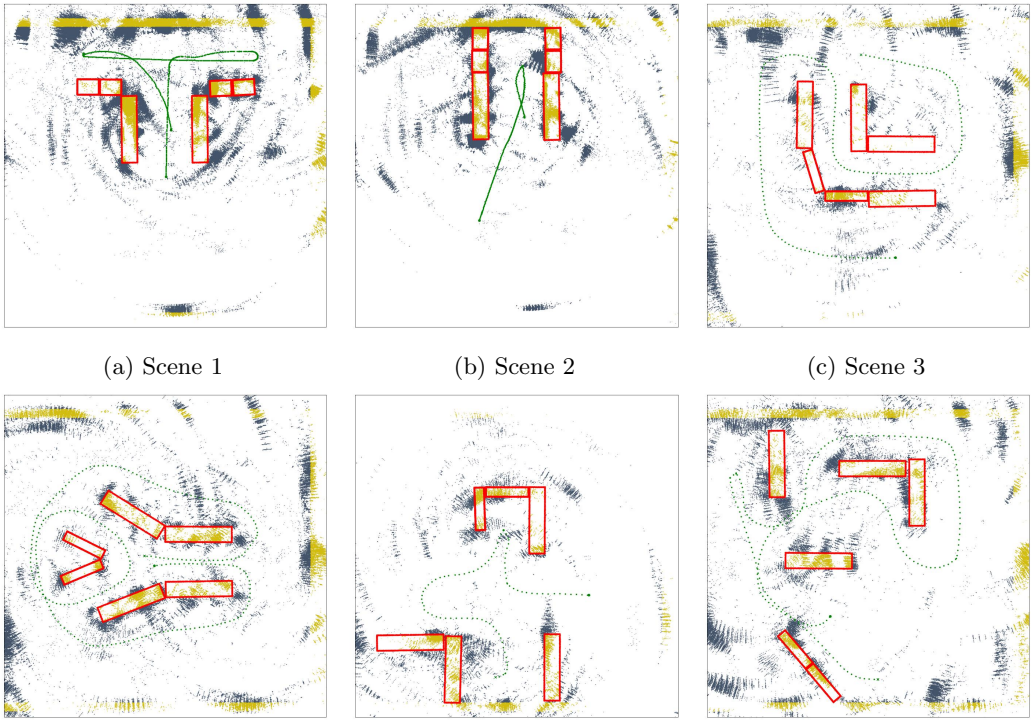
$$O_i = (x_{O_i}, y_{O_i}, \theta_{O_i}, w_{O_i}, h_{O_i}) \tag{4.1}$$

where x_{O_i} and y_{O_i} are the coordinates of the origin of the obstacle in the global coordinate system, θ_{O_i} denotes the orientation of the obstacle with respect to the global coordinate system, w_{O_i} and h_{O_i} are the width and the height of the object, respectively. The robot position and orientation in the global coordinate system is given by the tuple (x_R, y_R, θ_R). Notice that the origin of the robot's coordinate system $X_R Y_R$ is placed on the sensor. The point clouds acquired by the mmWave sensor as the robot moves in the environment are expressed in the robot's coordinate system.

Figure 4.4.3 shows the six scenarios collected in the MilliNoise dataset. For each of those scenarios, several collision-free paths were generated using Dijkstra's path planner, a state-of-the-art algorithm for global path planning. These paths were traversed by the robot using the Adaptive Monte Carlo Localization (AMCL) [87] while capturing PCs produced by the mmWave sensor.

Each instance of a robot traversing a given scenario is denoted as a *run*. Given the combined use of tracking and mmWave sensors, each run contains the scene's global information as well as the robot's individual "perspective". This is, each run contains: (i) the time-varying pose of the robot (x_R, y_R, θ_R) and the fixed position of the obstacles $(x_{O_i}, y_{O_i}, \theta_{O_i})$ in the global coordinates reference

system; (ii) the point clouds of the robot surroundings in the robot's coordinate system. Finally, notice that data have been captured at 2 Hz associating them to the current timestamp. Every point cloud captured from the mmWave sensor in this time window is collected under the same frame.



(a) Scene 1 (b) Scene 2 (c) Scene 3

(d) Scene 4 (e) Scene 5 (f) Scene 6

Figure 4.4.3: Example runs of each of the six scenes available in the MilliNoise dataset. Obstacles are shown in red. Clean (noise) points are shown in yellow (blue). The trajectory followed by the robot is shown with a dashed line.

4.4.2 Point Cloud Pre-Processing

In order to simplify the elaboration of the PCs, the sensor's firmware has been modified to consistently produce a fixed number of points per cloud frame. In particular, the number of points captured by the sensor for each frame was set at 200. Given the specific scenarios under consideration (i.e. short range), the number of points produced by the sensor for each cloud may fall below the desired target number.

Augmenting the desired number of points is a non-trivial task; the additional points introduced should align with the distribution of points actually captured by the sensor. It is worth noting that, due to the sparsity inherent in the PC generated by these sensors, each PC describes multiple regions, often associated to different objects in the scene. As a consequence, rather than considering

the distribution of the whole PC, the distribution of points within each smaller region is considered.

To this end, the *Agglomerative Hierarchical Clustering* (AHC) [88] algorithm is employed to reliably generate new points that keep the integrity of the original PC in terms of both shape and distribution. AHC is a hierarchical clustering method commonly used for data grouping. It follows a bottom-up search, iteratively "agglomerating" the closest data points. For each detected subgroup of points, the algorithm computes the centroid which is added to the PC with fixed fake velocity and intensity values. This allows one to differentiate between the original points and those artificially added to reach the desired total points in the PC.

4.4.3 Data Labeling

One of the main advantages of MilliNoise is the accurate point-wise annotation. The scene information provided by the tracking system, i.e. the poses of the obstacles and mobile robot, is used to annotate the point cloud acquired by the mmWave radar. To this end, each point cloud acquired by the mmWave sensor is *roto-translated* from the robot coordinate system to the global coordinate system. After an appropriate roto-translation and based on the known placement of obstacles, it is possible to automatically label each point as *real* or *noise*, as well as compute the distance of the point to the closest obstacle. Below, the labelling process is described.

Real or Noise Label: Given the obstacles and points roto-translated positions, if a point falls within an obstacle, it is labeled as *real*/*true*, whereas if it does not fall within the spatial region of any obstacle it is labeled as a *noise*/*false* point. For practical reasons, the walls of the room where the dataset was collected were considered as obstacles (real objects).

Distance to Object Label: Similarly, the distance between each point and its closest obstacle is computed. In particular, points falling within obstacles are assigned with a distance equal to zero. Hence, real points have a distance value equal to zero, while noise points will have increasingly higher distance values according to how far they are from the closest obstacle.

The distance annotation is particularly relevant in mmWave datasets since the boundary regions of objects pose several challenges. As shown in Figure 4.4.3, large clusters of noise tend to appear on the border of the objects. Their proximity to real objects means they are difficult to detect, possibly affecting navigation tasks. To tackle this issue, the distance label can be used to shift the denoising task from a binary classification problem into a regression one. This allows the system to learn that noise points at the borders of objects are not the same as noise points in the navigation routes. The idea is that the distance label can help the development of more robust navigation systems capable of distinguishing negligible false points at an object's border from false points in an otherwise empty region where the robot can move freely.

4.4.4 Dataset features

The MilliNoise dataset collects 49 individual runs captured in a diverse set of 6 real-world scenarios totalling around 58k frames for a duration of 8 hours of data collection. This enables to carry out in-depth exploration of temporal dynamics in the point clouds and its effect on noise points. Figure 4.4.3 shows an example run for each of the six considered scenarios, depicting true points and noise points, as well as the path followed by the mobile robot. In total, around 12 million individually labeled 3D points have been collected, with 60.15% of noise points (7 million points) and the remaining part of real points (5 million). Besides its (x_i, y_i, z_i) coordinates, each point is equipped with velocity and intensity information, which paves the way to analyses investigating the impact of these additional variables on the noise points.

4.4.5 Dataset Organization

In the following, more technical information on how MilliNoise is organized and how to use it are provided.

The dataset is organized in folders. Each folder contains data related to a single run. In particular, each folder contains the raw data and its post-elaborated version, which includes the labels and the distance to the closest obstacle.

For each run, the robot's position and captured point clouds are provided in the form of a JSON file with the following pair ¡*key, value*¿:

- `scene_ID`: the ID of the scene of that particular run;
- `timestamp`: the timestamp at which the given frame has been captured;
- `data_pose`: the list of robots' coordinates (x_R, y_R) in meters and orientation θ_R in radians;
- `data_mmwave`: this field contains data coming from the sensor. In particular, it contains the list of frames, in which each point is described by its coordinates in meters, intensity and radial velocity in m/s: $x, y, z, intensity, velocity$. The labeled dataset extends this field with the labels and the distance in meters from the closest obstacle $(x, y, z, intensity, velocity, l, d)$.

Regarding the point cloud augmentation process described in Section 4.4.2, it is worth noting that one can easily filter out added points. In fact, points added through the augmentation process are assigned with an intensity value equal to 255. Notice that such intensity value is never obtained by real data points acquired by the sensor.

The data folder also contains a text file describing the structure of the data and a file with information of each scene. The scenes' file contains the following information: the scene's ID, the

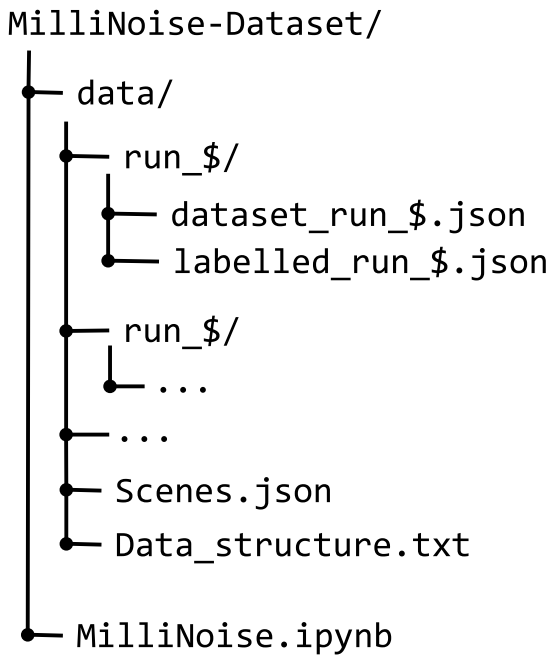


Figure 4.4.4: The directory structure of the dataset.

number of obstacles, the ID of each obstacle and the related position and sizes. A Jupyter notebook placed in the main folder provides examples on how to load, process, and visualize the mmWave data. Figure 4.4.4 depicts the resulting directory structure of MilliNoise.

4.4.6 Dataset Tools

In order to further improve usability and help bootstrapping applications, a few tools as examples on how to access and use the MilliNoise dataset are provided.

In particular, the `MilliNoise.ipynb` script provides several useful functions, the main ones being:

- `load_run`: it takes as input the directory containing the run one wants to load and returns the robot's poses and the mmWave frames in the form of list of lists;
- `apply_rototranslation`: it requires the mmWave frames and the list of robot's poses and returns a *numpy* array containing the mmWave frames rototranslated to build a map;
- `plot_scene`: this function 3D-plots the (numpy array) mmWave frames in input; optionally, it accepts the list of robot's poses to plot the trajectory, the list of points representing the obstacles, the points' sizes, the x and y limits to focus on a particular region of interest and the destination directory (and file name) to save the produced plot;

- `plot_2d_scene`: this function is analogous to *plot_scene* but projects the plot on the 2D XY plane;
- `load_train_test_data`: this function reads the provided path searching for mmWave data and returns the list of all frames from all runs in the directory; it optionally accepts a list to filter runs as train, test and validation data;
- `get_data_and_label`: it expects in input a list of mmWave labeled frames and returns the numpy arrays of frames and related points' labels. This function allows one to directly use data for training purposes.

4.5 Concluding Remarks

In this chapter, MilliNoise is presented, a labeled dataset with point clouds collected in indoor scenarios using a moving mmWave FMCW radar mounted on a mobile robot. MilliNoise is the first dataset collected with such an acquisition system in indoor scenarios. A distinctive feature of MilliNoise is that it provides an accurate point-wise labelling of each of its 12M points. In particular, label accuracy is only related to the accuracy of the motion tracking system, which is below a tenth of a millimeter. Furthermore, the sensor's firmware has been modified to provide a fixed number of points for each frame, enabling the employment of methodologies which require a fixed number of points for each frame. MilliNoise dataset can be a valuable tool for designing learning-based methods to denoise mmWave PCs, making these mmWave radars more efficient and useful in autonomous navigation systems and multimedia scenarios.

Chapter 5

GT-MilliNoise: Graph Transformer for Point-wise Denoising of Indoor Millimeter-Wave Point Clouds

5.1 Introduction

Building on the MilliNoise dataset presented in Chapter 3.6, this chapter proposes learning models for point-wise denoising of mmWave PCs.

mmWave radars are sensors operating with wavelengths in the order of millimetres. They are equipped with transmitter antennas, emitting signals, and receiver antennas, capturing their reflections. The processing of received signals enables the generation of a 3D PC that describes the environment, including information on the velocity, intensity, and orientation of the reflecting objects. Thanks to their operating frequencies, these sensors are significantly more robust in adverse environmental conditions, such as fog, dust, smoke, and rain [58, 62, 66, 67]. Despite such advantages, the PC data generated by mmWave sensors introduces some peculiar challenges. Specifically, *(i)* the presence of a large amount of noise, and *(ii)* the sparsity of points in each acquired frame. The higher level of noise is due to the wider wavelength of mmWave signal, which is highly susceptible to multi-path reflections. At the same time, the small antenna size and the low power signal of the mmWave lead to a lower signal-to-noise ratio (SNR), which further increases the noise and reduces the total number of points captured per frame. A typical mmWave PC consists of only $\sim$200 sparse points, as opposed to the tens of thousands found in LiDAR PC, and most of the points in mmWave PC are noise points. As a consequence, in a mmWave PC, shapes and objects are rarely understandable to the naked eye.

Given these unique challenges, it is not obvious if current PC processing approaches can be extended for the mmWave PCs. Most of the existing PCs learning architectures were developed for significantly denser and less noisy PCs generated by LiDAR sensors or RGB-D cameras. Such architectures have started to be adapted for mmWave PCs, but mainly focusing on PC level tasks such as action recognition [63, 78, 80] or broad-level object detection [77, 79]. Conversely, understanding how well current architectures from dense PCs behave with mmWave data for point level tasks (e.g., point-wise denoising) is highly overlooked and remains a gap in the literature.

This chapter proposes a first attempt at point-wise denoising tailored for the unique challenges of indoor mmWave PCs. To achieve this, the performance of several state-of-the-art approaches typically used for traditional PC processing are investigated. However, note that these have never been investigated for denoising highly sparse and noisy mmWave PCs. Specifically, the state-of-art point-based [41, 89], graph-based [90], and transformer-based [91] architectures have been implemented. The results obtained show that state-of-the-art methods exhibit limitations in exploring the local-to-global structures present in sparse PC and modelling the relations between points. To address these limitations, this chapter proposes GT, an architecture for mmWave PC denoising composed of two main cascading blocks:

1. A *Temporal* block, designed to explore the dynamic aspect of the input PC by learning how to distinguish between sporadic points (likely to be noise) and consistent points over time.

2. A *Geometric* block, which takes this temporal information into self-attention operation to form representative neighbourhoods able to explore the 3D space. From these neighbourhoods, a message-passing graph convolution captures relations between the points to learn spatial-temporal features for the denoising task.

The proposed GT is able to denoise mmWave scenes with an average 0.68 F1-score and 75% accuracy, corresponding to a 5% gain over the state-of-the-art. Beyond the quantitative gain in terms of accuracy (i.e., percentage of correct points), we also observe a significant gain in geometric metrics, which measure how similar the denoised scene is to the target ground truth. Simulation results demonstrate that the GT model better understands the 3D scene. As a last contribution, an ablation study is carried out to investigate how the input size and the inclusion of additional features (*i.e.*, velocity, intensity), as well as each component of the GT architecture, affect the model performance. This chapter ends by highlighting potential directions for improving research.

5.2 Background

This section describes FMCW radar PCs acquisition (Section 5.2.1) and provides a brief overview of the main classic approaches for PC processing (Section 5.2.2). Finally, it describes how these methods were adapted to mmWave PC data, specifically focusing on the denoising task (Section 5.2.3).

5.2.1 FMCW Radar and mmWave PCs

mmWave radars emit a set of signals (*chirp frame*), and based on their reflection, build a set of two-dimensional arrays describing the intensities of the received signals. By leveraging the Time-of-Flight (ToF), the displacement of the receiving antennas, and the Doppler effect, it is possible to extract a 3D PC representing the position of the objects, the intensity of its reflected signal and the radial relative velocity [69]. The interested reader is referred to [69] for an extensive overview of mmWave FMCW radar processing.

FMCW mmWave radars are gaining popularity for PC acquisition due to their superior ability to penetrate 3D space in low visibility conditions and their low price, making them easily accessible. They also adopted integrated antennas, Antenna-On-Package (AoP), resulting in significantly reduced production and retail costs, which helped increase the adoption of mmWave sensors in several research and industrial fields. However, the constraints on the signal and on the sensor's antennas limit the quality of the acquired PC representation. The mmWave wider wavelength is highly susceptible to multi-path interference and scattering. The interference effect is exacerbated in confined indoor spaces where multi-path reflections are formed more easily. Additionally, the low transmit power of mmWave radars, often necessary for regulatory compliance and power constraints, results in lower received signal power and, thus, a reduced SNR. This lower SNR contributes to both an increased amount of noise points and a reduction in the total number of acquired points. As a result of the above characteristics, PCs acquired from mmWave radars are significantly sparser and noisier compared to "traditional" PCs acquired by LiDAR sensors.

Outdoor PCs vs Indoor PCs: with respect to dense and outdoor point clouds, the first difference stems from the distance between the objects reflecting the antenna's signals. In fact, given the higher distance between objects (from at most a few meters in the indoor scenario to tens of meters in outdoor cases) the relation between objects is dramatically reduced. Not only that, in the outdoor case, the higher distance between objects prevents (or at the very least reduces) the chance of interfering reflections, increasing the overall quality of the point cloud. On the other hand, in indoor scenarios objects are much closer and the chance of signals interfering with each other is much higher, obtaining a much more noisier point cloud. This is further proved by the percentage of

noise points in the MilliNoise dataset (see Section 3). A second fundamental difference relies on the point-wise labelling: outdoor datasets often lack a point-wise accurate labelling methodology. The lack of accurate point-wise labels prevents the computation of any kind of loss function or metrics for both carrying out the neural network training and post-training evaluation.

Within this context, the importance of the MilliNoise dataset of accurately labelled mmWave indoor scenes for research on denoising methods becomes evident.

5.2.2 Point Clouds Processing

PCs are a set of unordered and irregular points. Processing such PCs to extract key information is challenging due to the lack of regular structures, which makes the usage of deep neural networks more involved. A common methodology to address this issue is to develop networks to handle raw PCs, without pre-processing steps (e.g., voxelization) that could obscure natural invariances of the data or introduce quantization errors. This approach was pioneered by PointNet [41] and PointNet++ [89] architectures, which proposed to process each point independently and aggregate the output via an invariant function. However, this point-based strategy fails to consider relationships between points when learning features. To address this issue, DGCNN [90] proposes to process the PC as a graph and to use a GNN to extract features by processing the edges between points. More recently, transformer-based architecture gained popularity for PC processing. The transformer considers the input PC as a fully connected graph. By considering all points, the Transformer can learn by itself the neighbourhood of each point for feature extraction. Transformer architectures, while powerful, may struggle to capture complex relationships between points in comparison with GNN, which are specifically designed to learn features from edges between points.

In [92], authors propose a multi-head self-attention mechanism based on learnable projections and rasterization and de-rasterization operations, composing a “cloud transform block”. The proposed architecture is evaluated on a dense point cloud for semantic segmentation, object classification, point cloud inpainting and image-based reconstruction. However, these tasks do not match the scope of this work, which is the point-wise classification of sparse indoor point clouds. In [93], authors present a point transformer block, which is used in combination with k-nearest neighbours (knn) layers and multilayer perceptrons. Later, in [94], the knn layer is replaced with a serialized neighbor mapping. Compared to our work, both architectures were evaluated on significantly denser point clouds for semantic segmentation and object classification tasks.

The approaches described above were originally designed for dense and relatively noise-free PCs from LiDAR sensors or RGB-D cameras. However, the ability of these models to extract key features from PCs for downstream tasks does not directly translate to mmWave sensors, due to the higher

levels of noise and sparsity in the mmWave PCs.

The Transformer architecture, while capable of capturing long-range dependencies between points, does not model these relationships and their complexity in the same way as message-passing GNN. Since it learns values V independently of the point neighbourhood, the self-attention operation is not as expressive as, f.i., graph message-passing convolution, which incorporates edge information by learning a message between two points.

In the next subsection, we describe these approaches, which have been applied specifically to mmWave PC processing, with a clear focus on the denoising task.

5.2.3 Related Work on mmWave Point Clouds Denoising

In current literature, mmWave PCs processing has been approached from two directions: 1) for human sensing applications and 2) for autonomous navigation applications. In the former, the developed methods process datasets [63, 77–80] consisting of sequences of human activities acquired using a *static* mmWave radar, with the final task of action recognition. By contrast, autonomous navigation methods, including ours, process data acquired from mmWave radar mounted on a *moving system*, such as a robot or vehicle. As the agent moves, the surrounding 3D space is captured as PC, which is then processed to facilitate navigation tasks [71].

Both applications share a common challenge of space-time feature extraction in highly sparse and noisy 3D environments, typical of mmWave acquisition systems. However, the type of noise and sparsity and the related impact on each task are different. Given the moving sensor and the environments with multiple objects, navigation PCs typically have more false/noise points compared to PCs for action recognition. This makes denoising for navigation more challenging, as it requires accurate labelling of each individual point. By contrast, action recognition only needs a single label for the whole point cloud sequence.

Despite this challenge, in the current literature, there is a lack of datasets of mmWave with accurate labels for denoising tasks. The lack of datasets is due to the difficulty of creating high-fidelity ground-truth annotations. Given this issue, the annotation process is usually simplified. The common approaches use a second modality as a reference (e.g. LiDAR) [95], are based on acquisition characteristics (e.g. Doppler effect) [86], or manually label selected points of interest [75]. However, these labelling strategies are usually incomplete and lack precision, with no guarantee of being an accurate ground truth (or label). Due to this lack of annotated data, most methods are measured based on some broad PC-level (or subset) detection task instead of accurate point-level metrics. In [96], a convolutional neural network (CNN) and PointNet++ are implemented for mmWave PC denoising of outdoor scenes. However, the authors only consider the points extremely close to the

Table 5.1: Comparison of mmWave Datasets for autonomous navigation tasks

Dataset	Runs	Frames (Points)	Environment	Annotation
nuScenes [71]	1000	1.3M (N/A)	Outdoor	Bounding Box
Astyx [72]	N/A	500 (N/A)	Outdoor	Object oriented
CARRADA [73]	N/A	12.6k (35M)	Outdoor	Bounding Box
OdomBeyondVision [76]	119	N/A (N/A)	Indoor	Not annotated
RadarScenes [74]	158	N/A (N/A)	Outdoor	Point-Wise; Labels for 10 Objects class + 'Others' class
Radar-Ghost [59]	111	N/A (35M)	Outdoor	Point-Wise; Labels for 5 Objects class + 'Background' class
APEIRON (ours)	49	58k (11.6M)	Indoor	Point-Wise sub-mm accuracy; Labels for True or False class; and distance-to-object label

sensor. In [59, 97] a Similarity Group Proposal Network (SGPN) [98] and a PointNet++ [89] were used, respectively. However, only a selection of points was labelled in the dataset [75] and used in these works. A large number of points were simply labelled as "background" and ignored. A complete denoising of the mmWave PCs was done in [95]. To make this possible, the authors create an algorithm to annotate the mmWave PCs dataset using a secondary LiDAR sensor. Given the fully annotated data, they implemented a simple PointNet architecture, achieving a 0.73 F1-score. However, all the above works process PCs acquired from outdoor scenes. To the best of our knowledge, our work is the first to explore *indoor navigation* PC denoising.

As seen in Section 5.2.2, most proposed architectures for mmWave denoising are either simple PointNet and PointNet++. These are usually modified, f.i. replacing the sampling step with mean shift clustering [99] or adding more processing blocks [95, 100]. However, the backbone of PointNet point-based operations remains constant. Looking more broadly at processing PC networks, there are several works [42, 58, 66, 67] outperforming point-based architectures with more advanced architectures such as graph-based or transformer-based deep neural networks. However, those works do not focus on point-wise denoising tasks. They are also usually considered in multi-modal settings, processing a combination of mmWave data with 2D RGB images or LiDAR. Within this context, our MilliNoise dataset provides the unique opportunity to benchmark existing methods and develop new ones for point-wise denoising tasks. Illustrated in Table 5.1, MilliNoise is the first mmWave PC dataset of *labelled indoor* scenes with a sub-millimeter accuracy and guaranteed to be correct. In this paper, we benchmark the most popular point-based architectures, identify key limitations, and propose a novel architecture, namely GT, outperforming existing ones in the denoising task.

5.3 mmWave Point Cloud Denoising

This section proposes the GT approach for mmWave PC denoising. It starts with the definition of the denoising problem and then presents the GT method, discussing first its strategy to learn features from mmWave PCs and later its architecture details.

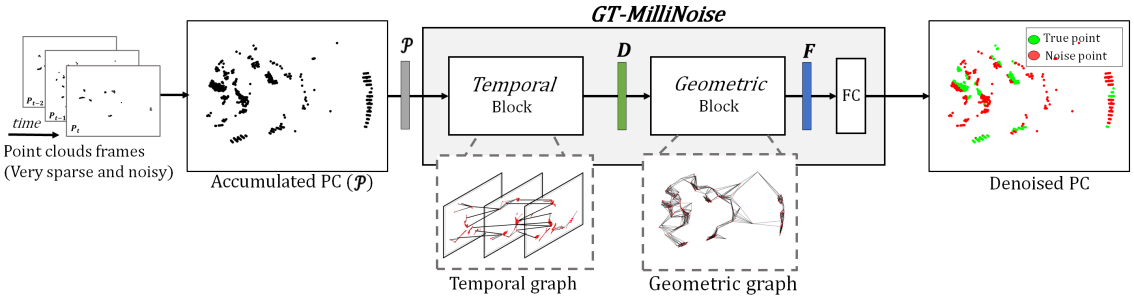


Figure 5.3.1: Proposed pipeline architecture for PC denoising composed of two main blocks: *Temporal* and *Geometric* and example how on the PC is converted to graph in each block

Problem Formulation

The denoising problem is tackled as a binary classification task. Given an input PC frame $P_t = \{p_1, \cdots, p_M\}$ defined as a set of points $p_i = \{x, y, z\} \in \mathbf{R}^3$, the model outputs the probability of the point being true/real or false/noise. This score is then used to calculate the *Binary Cross Entropy* (BCE) used as training loss.

5.3.1 Proposed GT Approach

Our goal is to extract features of mmWave PCs, which is made challenging by the sparsity and the high amount of noise points in the scene. Given these challenges, the GT strategy is to:

1. increase the input data to the model accumulating a number of consequent frames (*frame accumulation*);
2. exploit the sparsity of each PC frame to learn the point's temporal behaviour (*temporal processing*);
3. employ a point-wise attention mechanism to build representative neighbourhoods for feature extraction and to model the large amount of noisy and sporadic points in the 3D scene (*geometrical processing*).

The GT approach starts by taking an accumulating window of PC frames as the model input. Formally, the *Accumulated Point Cloud* $\mathcal{P}(t, W)$, at a time-stamp t, is defined as the PC obtained by accumulating a window of W sequential PC frames together as follows: $\mathcal{P}(t, W) = (P_{t-W+1}; \cdots; P_t) \in \mathbf{R}^{N \times 3}$ with $N = WM$ points, where each frame is associated with a timestamp from the vector $T = \{t - W + 1, \ldots, t\} \in \mathbf{R}^{W \times 1}$. Intuitively, accumulating PCs from multiple time steps not only increases the amount of geometric information available but also provides a temporal

description of the scene. To leverage both these temporal and geometric features, the augmented PC is processed by two main processing blocks: (1) a *Temporal* block and (2) a *Geometric* block.

The goal of the *Temporal* block is to extract the temporal information of the scene from the input accumulated PC. However, extracting such information from PC data is challenging since there is no explicit point-to-point correspondence over time. This challenge is even exacerbated in the case of mmWave PCs due to their high sparsity. Nonetheless, the same sparseness of data can be leveraged to extract valuable temporal information. In fact, it is possible to exploit the sparsity to capture the notion of *point temporal consistency*. Specifically, a point is considered "*consistent*" over time when its 3D position is regularly occupied at other time steps. In contrast, "*sporadic*" points appear abruptly in empty areas and vanish almost instantly. Figure 5.3.2 presents an example of a sporadic (in yellow) and a consistent (in blue) point in time. It is known that mmWave PCs have a significantly higher amount of sporadic points compared to traditional point clouds. Then, the intuition is that while consistent points could be associated to actual objects, the sporadic points are more likely due to noise and are generally not critical for understanding the scene's geometry. The *Temporal* block is a set of network layers designed to extract this temporal aspect and is presented in Section 5.3.2.

Concerning the *Geometric* block, its goal is to capture the geometric structures within the PC data. In PC literature, it has been shown that such structures can be captured by aggregating information from geometric neighbourhoods. However, such a process is not as straightforward for mmWave PCs. The common assumptions for traditional PC do not necessarily apply to mmWave PCs. For instance, since in mmWave PCs false points can still fall close to true points, the ones in close proximity in the geometric space do not necessarily belong to the same latent object space. This makes it very challenging to define effective neighbourhoods and, within those neighbourhoods, to aggregate information, given that a significant percentage of the points are false and unreliable. To address these issues, the *Geometric* block employs a self-attention mechanism to dynamically find informative neighbourhoods from each point. For this operation, the attention mechanism leverages the temporal features learned in the *Temporal* block, which identifies the sporadic and unreliable points for understanding the scene's geometry. Using the learned attention, the *Geometric* block constructs a graph representation, which allows us to extract features from informative neighbourhoods and model complex relationships between points connected by edges. The *Geometric* block is explained in detail in Section 5.3.2.

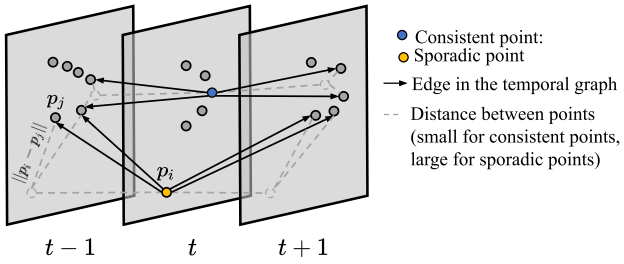


Figure 5.3.2: Example of how the edges are built in the temporal graph $\mathcal{G}_\mathcal{T}$ processed by the *Temporal* block. The point in blue (yellow) is an example of a consistent (sporadic) point in time.

5.3.2 GT Architecture

In the following, the proposed GT architecture for PC denoising, depicted in Figure 5.3.1, is described. GT takes in input the accumulated point cloud $\mathcal{P}$. Then, a first *Temporal* block builds a graph from the accumulated point cloud frames, from which it learns dynamic features $D = \{d_1, ..., d_N\} \in \mathbf{R}^{N \times C_D}$. The learned dynamic features are then sent along with the input PC to a *Geometric* block that builds a *geometric* graph to learn geometric features $F = \{f_1, ..., f_N\} \in \mathbf{R}^{N \times C_F}$. In the last step, the learned geometric features are concatenated to learn a global representation and are processed by multiple fully connected (FC) layers, which outputs a classification score for whether a point is a real point or a noise point.

Temporal Block

At its core, the *Temporal* block is a message-passing GNN [101]. However, unlike conventional GNNs that aggregate features from spatial neighbourhoods, the *Temporal* block aggregates features from temporal neighbourhoods. To this end, the *Temporal* block starts by building a temporal graph $\mathcal{G}_\mathcal{T} = (\mathcal{V}, \mathcal{E})$ by considering each point in $\mathcal{P}$ as a node in the graph and building edges between points across time. Specifically, for each point p_i of the P_{t_α} frame at time step t_α, an edge is built to its k geometrically closest neighbours in the other frames (P_{t_β} with $t_\alpha \neq t_\beta$). An example of temporal graph $\mathcal{G}_\mathcal{T}$ is depicted in Figure 5.3.2.

After the temporal graph is built, the *Temporal* block performs a message-passing convolution to aggregate information from the temporal neighbourhoods. More specifically, for the α-th frame, for each i-th point of the l-th layer, the *Temporal* block concatenates: *(i)* the i-th point temporal feature d_i^{l-1}; *(ii)* the temporal feature d_j^{l-1} from its j-th neighbourhood, from time-step t_β, *(iii)* the geometric distance $\|p_i - p_j\|$ and *(iv)* the time displacement $\|t_\alpha - t_\beta\|$, between point i and j. This large concatenation is processed by a learnable layer $\Theta_\mathcal{T}^l$ to learn the message m_{ij}^l, modelling the relation between the two points.

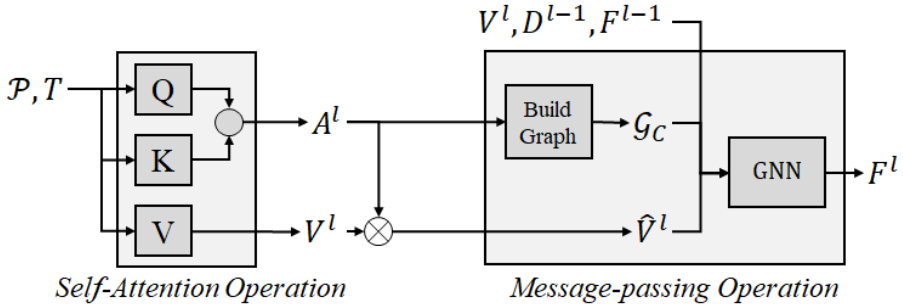


Figure 5.3.3: Schematic of a *Geometric* block layer.

$$m_{ij}^l = \Theta_{\mathcal{T}}^l\left(d_i^{l-1}; d_j^{l-1}; \|p_i - p_j\|; \|t_\alpha - t_\beta\|\right) \tag{5.1}$$

Next, for each point i, its neighborhood messages m_{ij}^l are aggregated via max pooling $\bigoplus$ to compute the temporal feature $d_i^l \in D^l$ as follows.

$$d_i^l = \bigoplus_{j \in \mathcal{N}_{\mathcal{T}i}} \{m_{ij}^l\} \tag{5.2}$$

After the *Temporal* module extracts the temporal features D, those features are combined with the original stacked PC $\mathcal{P}$ and feed them to the *Geometric* block as shown in Figure 5.3.1. Given the inclusion of temporal features, the *Geometric* block performs a neighbourhood aggregation with the prior notion of each point's temporal consistency. In Section 5.5.5, an investigation on the impact of the *Temporal* block on the overall GT performance.

Geometric Block

The *Geometric* block takes the accumulated PC $\mathcal{P}$ and temporal features D as input and outputs geometric features F for each point, as depicted in Figure 5.3.3. Each layer l of the *Geometric* block is a combination of a self-attention operation and a message-passing operation. In each layer, it is first learnt the neighbourhoods using the self-attention mechanism. Then, using the learned attention, it is built a *geometric* graph $\mathcal{G}_c$, from where a message-passing operation extracts geometric features. In the following each operation is explained in details.

Self-Attention operation: The goal of the self-attention operation is to learn how each point relates to another. Thus, the self-attention mechanism begins by learning *queries* ($Q \in \mathbf{R}^{N \times C_Q}$), *keys* ($K \in \mathbf{R}^{N \times C_K}$, with $C_Q = C_K$), and *values* ($V \in \mathbf{R}^{N \times C_V}$), as the linear projection of the concatenation of the input accumulatedPC. An *attention matrix* ($A \in \mathbf{R}^{N \times N}$) is computed by taking the dot product of the query vectors and the key vectors and then dividing by the square

root of the dimension of the query vectors. The end result is an attention matrix containing a scalar value representing the *point attention* with respect to all the other points.

Lastly, the refined values $\hat{V} \in \mathbf{R}^{N \times C_V}$ are learnt by performing a weighted sum of value vectors, where the weights are determined by the attention matrix.

$$\hat{V} = AV, \quad A = \text{softmax}\left(\frac{QK^T}{\sqrt{C_K}}\right) \tag{5.3}$$

This weighted aggregation is the core of the Transformer architecture [91]. However, since it learns values V independently of the point neighbourhood, this operation is not as expressive as graph message-passing convolution, which incorporates edge information by learning a message between two points. As such, to model complex relations between points, a graph message-passing convolution is employed after the self-attention mechanism.

To this end, a *geometric* graph $\mathcal{G}_c$ is built based on the learned attention values. Specifically, for each point i, it is built an edge connecting it to the top k points with the highest attention matrix A_i to point i. In contrast to the temporal graph, where edges only connect points across a single time step, the geometric graph has learned attention values for every point-to-point relationship in the PC. This grants the model the flexibility to dynamically define informative neighbourhoods for feature aggregation among all possible combinations. Furthermore, for each point, this attention-selected neighbourhood varies at each layer of the *Geometric* block, enabling the network to explore the 3D space at varying scales (from local to global) to capture the geometric structure within the data.
Message-passing operation: Given the geometric graph, the network performs a message-passing convolution by processing the concatenation of: *(i)* the values v_i^l, *(ii)* the refined values $\hat{v}_i^l$, *(iii)* the values related to the neighbors v_j^l and *(iv)* the learned attention scalar between points A_{ij}.

$$m_{i,j}^l = \Theta_m^l\left(v_i^l; \hat{v}_i^l; v_j^l; A_{ij}^l\right) \tag{5.4}$$

The geometric features (f_i^l) for each point are then computed as the message aggregation.

$$f_i^l = \bigoplus_{j \in \mathcal{N}_i} \left\{m_{ij}^l\right\} \tag{5.5}$$

By combining the self-attention and message-passing operations, at each layer $l-th$ the *Geometric* block takes advantage of the best of both approaches. The self-attention enables the model to explore the 3D neighbourhoods to capture local-to-global geometric structures (later shown in Section 5.5.4), and the message-passing enables the model to learn complex relations between points.

Table 5.2: The considered train and test splits.

K-Fold:	Training Scenarios	Test Scenarios
Fold-1-6	*All* [1-6]	*All* [1-6] (19 runs)
Fold-123	[4,5,6]	[1,2,3] (10 runs)
Fold-4	[1,2,3,5,6]	4 (10 runs)
Fold-5	[1,2,3,4,6]	5 (6 runs)

5.4 Implementation

This section describes how the model is implemented and the experimental conditions.

5.4.1 Training

We tackle the denoising problem as a binary classification task. For every point $P_i \in \mathcal{P}$, the model learns the probability $p_\theta(Y_i)$ of a point being a true point ($Y_i = 1$) or being a false point ($Y_i = 0$). Therefore, the proposed GT and benchmarking methods are trained using the common *Binary Cross Entropy* (BCE) as a loss function, as follows.

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} Y_i \log(p_\theta(Y_i)) + (1 - Y_i)\log(1 - p_\theta(Y_i)) \tag{5.6}$$

where Y_i is the ground-truth label for the point i.

K-Fold Train/Test Splits: To provide a robust evaluation of our proposed method, we employ a cross-validation approach to split the dataset into multiple train/validation/test sets. We then train the same model on each of the considered data splits and average their performance. The splits are presented in Table 5.2. The MilliNoise dataset consists of 6 scenarios, with each scenario being one unique obstacles course navigated by the robot. Then, for each scenario, different runs have been considered, generating multiple and diverse robot paths per scenario. The splits in Table 5.2 have been created in order to evaluate the model performance across different scenarios. For example, in the split named *Fold-4*, the model is trained on runs from scenarios $1, 2, 3, 5, 6$ and evaluated on runs from scenario 4. The exception to this rule is the split named *Fold*-1-6. In this split, the same scenarios are shared for the train/validation/test sets. It is worth mentioning that even when one scenario is shared across training, validation and testing, different runs are used for different sets, preventing any data leakage. This is because the PCs are captured from the robot's reference frame, and each individual run follows a particular trajectory within the scenario, making each run unique. As a consequence, different runs across train/validation/test sets in the *Fold*-1-6 imply that the model is evaluated on previously unseen data.

5.4.2 Evaluation Metrics

To evaluate the performance of our models on the denoising task, we consider BCE, accuracy, recall, and F1-score on the test dataset. While the above metrics are widely used in classification tasks, they are not sufficient to measure the quality of the denoised PC. Specifically, these metrics are averaged across all points. This means that regions with a high density of points have a significantly higher impact on the final metric. However, for tasks such as robot navigation, *not all points have the same importance.* While it is still desirable to have a high percentage of correctly classified points, it is more critical to correctly classify the points that actually describe obstacles in order to avoid them. Therefore, we consider additional *geometric metrics*, in order to compare the geometry of the growth-truth PC with the estimated one.

Geometric metrics for denoising quality: Given a complete run of the robot across one scenario, we define P^{true} as the set of points in PC that are true objects, and $\hat{P}^{true}$ as the point classified by the model as real by the model, as follows.

$$P^{true} = \{p_i \in P_t \quad | \quad label(p_i) = \text{true point}\} \tag{5.7}$$

Hence, P^{true} is the true representation of the scenario, and $\hat{P}^{true}$ is what the model perceives as the scenario. Our goal is to measure the geometric difference between these two PCs. The smaller the differences, the better the model's understanding of the scene. To this end, we use both the Chamfer distance (CD) [102] and Earth's Moving Distance (EMD) [103]. These metrics are widely used in the literature for PC comparison and are defined as follows.

Chamfer distance (CD): The CD measures the distance between each point in the predicted PC and its closest target point in the reference PC, and vice-versa.

$$CD(P^{true}, \hat{P}^{true}) = \frac{1}{n} \sum_{p_i \in P} \min_{\hat{p_i} \in \hat{P}} ||p_i - \hat{p}_i||^2 + \frac{1}{n} \sum_{p \in \hat{P}} \min_{p_i \in P} ||\hat{p}_i - p_i||^2 \tag{5.8}$$

Earth's moving distance (EMD): The EMD between two distributions (or PCs in this case) is proportional to the minimum cost required to change one distribution into the other. More formally, the EMD solves an optimization problem by finding the optimal point-wise bijection mapping θ between two PCs, namely $\theta : P^{true} \rightarrow \hat{P}^{true}$. The EMD distance is then given by the distance of the points at both ends of this mapping, as follows:

$$EMD(P^{true}, \hat{P}^{true}) = \min_{\theta : P^{true} \rightarrow \hat{P}^{true}} \sum_{p_i \in P^{true}} ||p_i - \theta(p_i)||^2. \tag{5.9}$$

Notice that the EMD requires the PCs in input to share the same size. Since the number of true points in the ground truth PC and the true points identified by the model can be different, we downsampled both the complete PC runs to $1,000$ points using FPS. We found this sampling value to provide a good overall representation of the PC geometry, simplifying the optimization task without affecting the final score.

It is worth noting that the EMD and CD were only used during the evaluation phase and not during training since their implementation as loss function is not trivial. In particular, these metrics are only applied to a subset of the input PC (P^{true}, the true points). As such, these metrics are very susceptible to the number of true points in the input PC. However, in PCs from the MilliNoise, the number of true points varies greatly from frame to frame, with cases lacking true points, causing the geometric loss values to be very volatile (high variation) during training.

5.4.3 Bechmarking Methods

One of the contributions of our work is the benchmarking of classic PC processing methods designed for traditional PCs, on mmWave PCs from the MilliNoise dataset. For this analysis, we selected architectures representing the main PC processing strategies, which are point-based (PointNets), graph-based (DGCNN) and attention-based (Transformer):

PointNet: [41]: PointNet is able to learn directly from PC data by processing each point independently via a shared learnable function and aggregating the output.

PointNet++: [89]: extends PointNet point-based operation to hierarchical neighbourhoods by sampling the PC between layers. The hierarchical neighbourhoods are defined in an *ad-hoc* manner by selecting the downsampling factor. PointNet and PointNet++ are implemented following the original papers [41, 89].

Dynamic Graph Convolutional Network (DGCNN): [90] DGCNN is a GNN that learns features by processing the PC as a graph. For each point, the DGCNN learns features by aggregating information from the point's neighbourhood, defined by the graph edges. The graph is recomputed at each layer based on feature similarity learned in the previous one. The DGCNN is implemented following the original paper [90].

Transformer [91]: While the first Transformer architecture was designed for text-data processing, its ability to learn unstructured data quickly lead to it's adaptation for PC processing by multiple works. For the sake of generality, during benchmarking, we implemented a "*vanilla*" Transformer architecture. Our benchmarking Transformer is composed of four self-attention layers in parallel as described in the *Geometric* block (Section 5.3.2).

5.4.4 GT Architecture Details

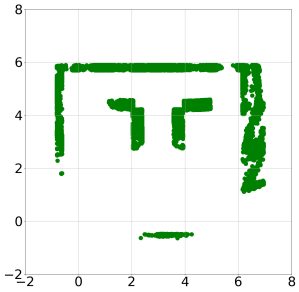
For all the considered architectures, a batch size of 32 is selected, the point neighbourhood is set to 8 points ($k = 8$), and hyper-parameters are fine-tuned for each architecture. Batch normalization is applied at the end of each layer. The models are trained with a learning rate of either 0.01 or 0.001 using ADAM optimizer, depending on architecture and accumulation window W. The GT is implemented with four layers, each layer learning features with 64 dimensions.

5.5 Results

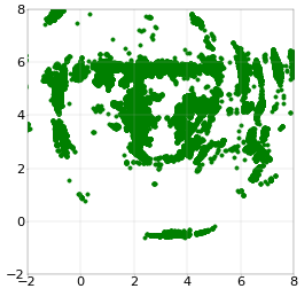
This section presents and analyzes the results obtained from the architectures on denoising the MilliNoise dataset. Before carrying out an in-depth analysis, in Section 5.5.1 we first highlight how classical classification metrics and geometrical metrics can lead to conflicting results. Next, in Sections 5.5.2 and 5.5.3, we provide a comprehensive analysis to shed more light on the advantages and drawbacks of the considered architectures when dealing with mmWave PCs. Section 5.5.4 investigates how each considered architecture explores the 3D space to extract features. Section 5.5.5 presents an ablation study to analyze the impact of the different aspects of the input data as well as the aspects of the model architecture on performance. Finally, Section 5.5.7 presents the limitations of this work and discusses possible future research directions.

5.5.1 Understating of Accuracy and Geometric Metrics

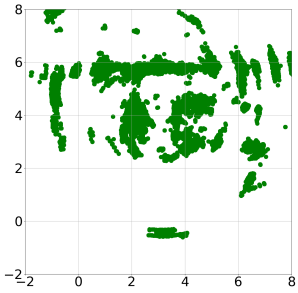
In the Figures 5.5.1 and 5.5.2, we show the visual denoising results of the different models for two runs of the robot. The points are roto-translated to a global reference frame to facilitate understanding, and the caption reports the accuracy and EMD value for each model. At a quick glance, it can be seen that examples with the best (highest) accuracy do not necessarily have the best (lowest) EMD values. An example of this discrepancy can be seen in Figure 5.5.1, between PointNet and PointNet++, and in Figure 5.5.2, between PointNet and DGCNN. In the former instance, the PointNet (Figure 5.5.1.a) was able to denoise a run in scenario 1 with 66% accuracy. In comparison, PointNet++ (Figure 5.5.1.b) has 61%, and the Transformer (Figure 5.5.1.e) has 67% accuracy. However, despite the lower accuracy, the PointNet++ and Transformer visualizations are more similar to the ground truth scene than PointNet visualization. This "geometric similarity" is instead better captured by the EMD metric, where PointNet++ and Transformer have significantly lower EMD compared to PointNet. This mismatch between metrics occurs because the accuracy measures how many individual points are correct and, as such, is biased by high-density regions. On



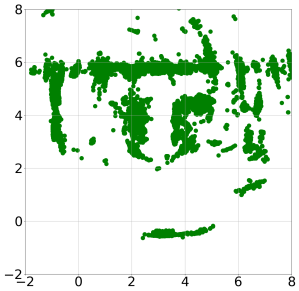
(a) Ground Truth - Real Points



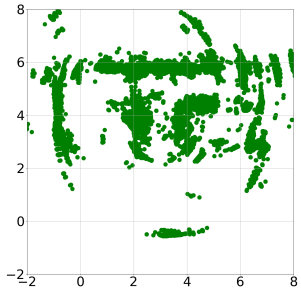
(b) PointNet (Acc: 66% , EMD: 0.65)



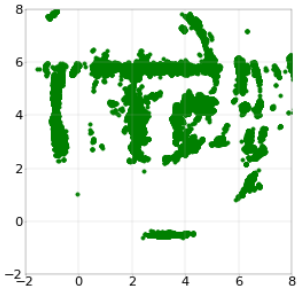
(c) PointNet ++ (Acc: 61% , EMD: 0.52)



(d) DGCNN (Acc: 68% , EMD: 0.50)

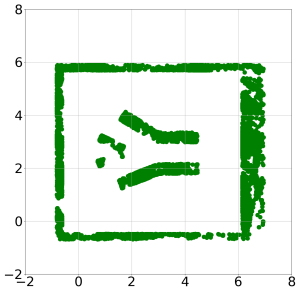


(e) Transformer (Acc: 67%, EMD: 0.53)

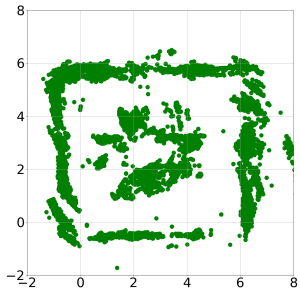


(f) GT (Acc: 69% , EMD: 0.48)

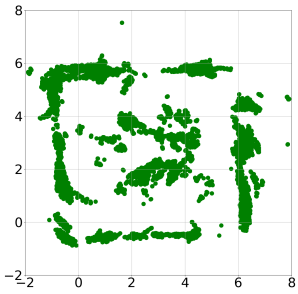
Figure 5.5.1: Denoising results of Scene 1 (*Fold-123* run 4).



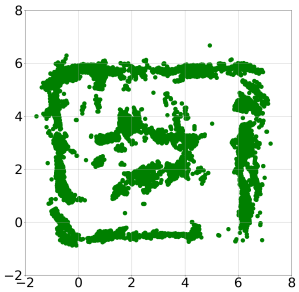
(a) Ground Truth - Real Points



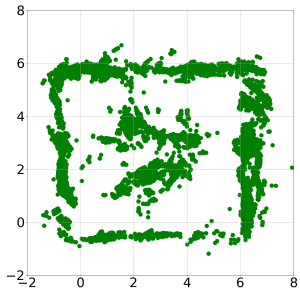
(b) PointNet (Acc: 76% , EMD: 0.61)



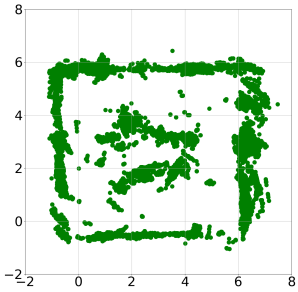
(c) PointNet++ (Acc: 72% , EMD: 0.50)



(d) DGCNN (Acc: 75% , EMD: 0.44)



(e) Transformer (Acc: 78% , EMD: 0.41)



(f) GT (Acc: 81%, EMD: 0.33)

Figure 5.5.2: Denoising results of Scene 4 (*Fold-4* run 70).

Table 5.3: Performance results averaged across folds

Architecture	BCE ↓	Acc. ↑	Recall ↑	F1 ↑	EMD ↓	CD ↓
PointNet	0.661	0.710	0.585	0.613	0.434	0.221
PointNet++	0.640	0.667	0.464	0.520	0.499	0.243
DGCNN	0.656	0.708	0.600	0.618	0.427	0.214
Transformer	0.578	0.700	0.604	0.613	0.418	0.210
GT (ours)	**0.550**	**0.750**	**0.674**	**0.681**	**0.360**	**0.193**

the other hand, the EMD measures how similar the predicted PC is to the ground truth and, hence, how well the model understood the overall scene.

5.5.2 Overall Results

The visual results from Figures 5.5.1 and 5.5.2 clearly show the proposed GT is better at denoising the scene compared to benchmarking methods. While the state-of-the-art approaches can capture the overall representation of the scenario (walls and objects), the GT are clearly closer to the ground truth and are visually cleaner. Specifically, the object's boundaries are "sharper," and the paths where the robot can move safely are mostly unobstructed.

The GT superior performance is validated by the quantitative results obtained. Table 5.3 presents average results across train/test splits, given an accumulating frame window length of $W = 12$. The results show that the proposed GT architecture consistently outperforms all the remaining architectures, achieving 75% accuracy and an F1-Score of 0.68. In comparison, the state-of-the-art architectures PointNet, DGCNN, and Transformer achieved an average$\sim$ 70% accuracy and F1-score of ~ 0.61. PointNet++ was an exception, achieving an underwhelming 64% accuracy. Considering the difficulty of the task, the 70% accuracy achieved by most state-of-the-art architectures is a positive result. These performance show these architectures originally designed for traditional PCs can be applied to indoor mmWave PCs for early results.

However, despite the success of the state-of-art approaches, the GT still represents a substantial improvement over them. Although the GT accuracy gain might seem modest ($\sim 5\%$), the GT has an overall better understanding of the 3D scene. This better understanding is represented by the significantly lower EMD of 0.36, in comparison with the benchmarking methods EMD of ~ 0.43. This better understanding of the scene translates visually to cleaner object shapes with less noise and distortion, as can be seen in the denoising scenes from Figures 5.5.1 and 5.5.2. Additionally, the GT lower BCE value indicates that even misclassified points are associated with lower confidence, enabling the recognition of uncertain regions while maintaining high accuracy for the ones with high confidence.

Table 5.4: Performance results on K-fold data split.

Evaluation of Scenes: 1,2,3,4,5,6 Fold-1-6							Evaluation of Scene: 1,2,3 Fold-123					
Architecture	BCE ↓	Acc. ↑	Recall ↑	F1 ↑	EMD ↓	CD ↓	BCE ↓	Acc. ↑	Recall ↑	F1 ↑	EMD ↓	CD ↓
PointNet	0.536	0.740	0.623	0.651	0.387	0.222	0.712	0.694	0.562	0.592	0.452	0.237
PointNet++	0.619	0.659	0.350	0.445	0.580	0.255	0.691	0.650	0.541	0.550	0.536	0.265
DGCNN	0.562	0.712	0.625	0.628	0.444	0.226	0.697	0.713	0.572	0.612	0.387	0.213
Transformer	0.499	0.750	**0.725**	0.693	0.405	0.209	0.635	0.654	0.537	0.551	0.377	0.209
GT (ours)	**0.480**	**0.794**	0.695	**0.725**	**0.309**	**0.179**	**0.593**	**0.725**	**0.695**	**0.666**	**0.328**	**0.197**
Evaluation of Scene: 4 Fold-4							Evaluation of Scene: 5 Fold-5					
Architecture	BCE ↓	Acc. ↑	Recall ↑	F1 ↑	EMD ↓	CD ↓	BCE ↓	Acc. ↑	Recall ↑	F1 ↑	EMD ↓	CD ↓
PointNet	0.659	0.734	0.629	0.629	0.490	0.214	0.736	0.670	0.525	0.580	0.406	0.211
PointNet++	0.592	0.693	0.505	0.542	0.478	0.247	0.657	0.664	0.460	0.543	0.402	0.206
DGCNN	0.652	0.725	0.620	0.617	0.447	0.214	0.713	0.683	0.584	0.615	0.430	0.202
Transformer	0.546	0.733	0.584	0.611	0.450	0.214	0.646	0.662	0.576	0.596	0.439	0.208
GT (ours)	**0.542**	**0.761**	**0.644**	**0.660**	**0.443**	**0.198**	**0.583**	**0.720**	**0.662**	**0.672**	**0.361**	**0.196**

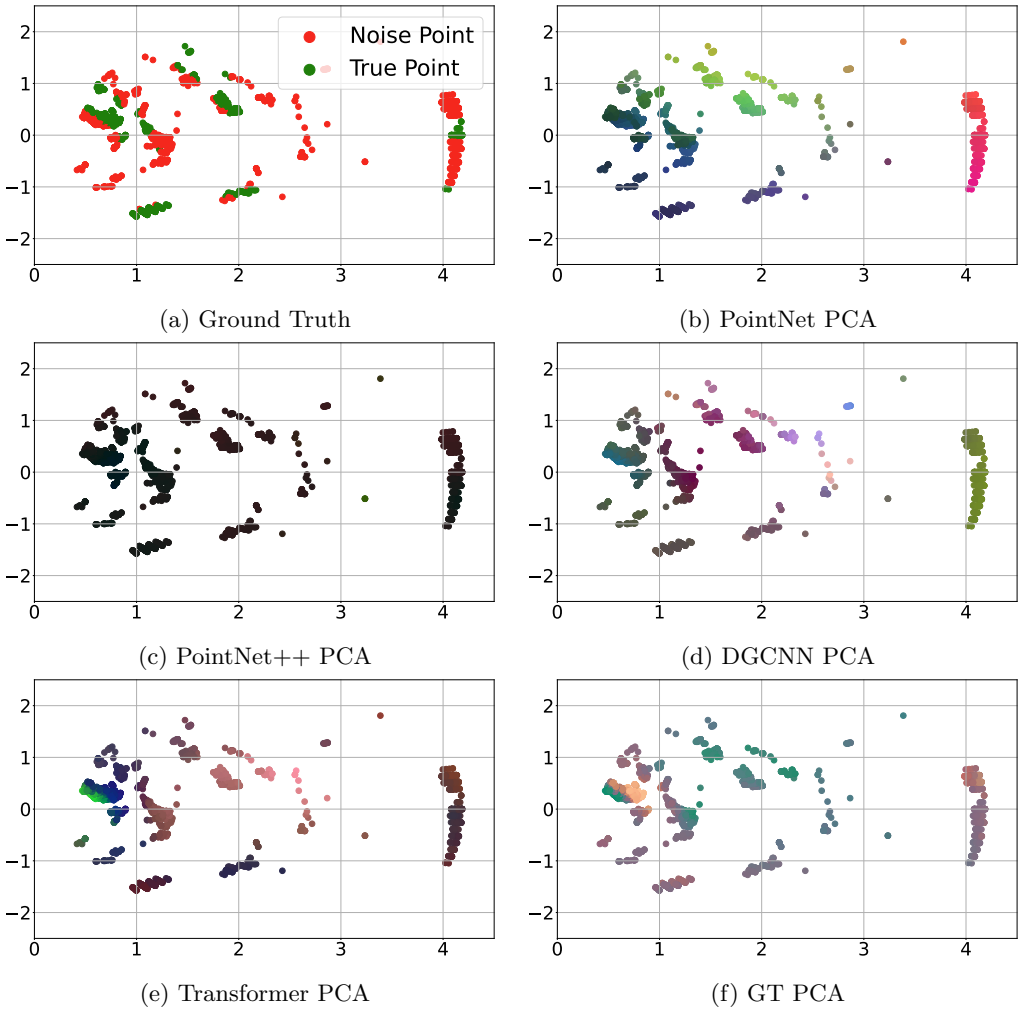
5.5.3 Generalization to unseen scenarios - K-Folds Results

To understand how well the model can generalize to unseen scenarios, we investigate the performance for the different train/test splits in Table 5.4. The *Fold-1-6* results are relevant in applications where the robot navigates in "familiar" scenarios, such as a factory robot trained on its specific floors. Under these train/test conditions, the GT achieves a high accuracy equal to $\sim 80\%$. To note, the test scenarios are different trajectories than the training one, ensuring the model is evaluated on unseen data. The results from the *Fold-123; Fold-4; Fold-5* are relevant for applications where the robot operates in scenarios completely different from its training data. Even in this more challenging setting, our model achieved positive results with denoising accuracy ranging from 72% to 76%. These results demonstrate the GT ability to generalize to unseen scenarios, indicating its potential for broader real-world applications.

5.5.4 3D Space Exploration Discussion

To better understand the obtained results and how traditional PC approaches handle mmWave data, this section investigates how each architecture explores the 3D space to extract features. From this investigation, we highlight each architecture's main benefits and shortcomings when dealing with mmWave PCs.

To this end, for each model, we project the high-dimensional features from the last layer of the neural netwok – before being converted into classification scores – into a low-dimensional domain using Principal Component Analysis (PCA). Given a PC as input, Figure 5.5.3 shows for each architecture the learned PCA features as point colour. Figure 5.5.4 shows the points classified as true by each model. Figure 5.5.5 shows a zoom-in of the PCA in a particular region of interest. In the figures, similar colours represent points with similar high-dimensional features. The ground-truth



(a) Ground Truth

(b) PointNet PCA

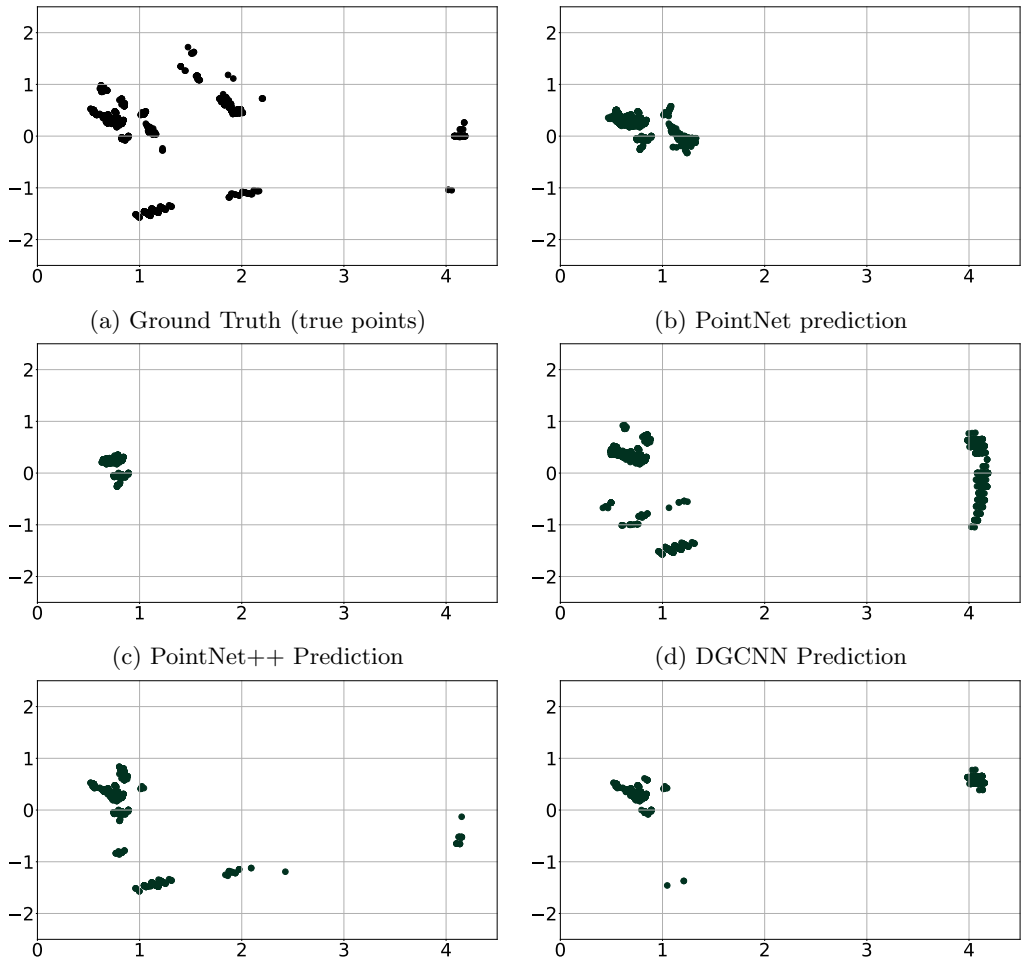
(c) PointNet++ PCA

(d) DGCNN PCA

(e) Transformer PCA

(f) GT PCA

Figure 5.5.3: Features PCA of each models (PCs with $W = 12$ and plotted from the robot perspective).



(a) Ground Truth (true points)

(b) PointNet prediction

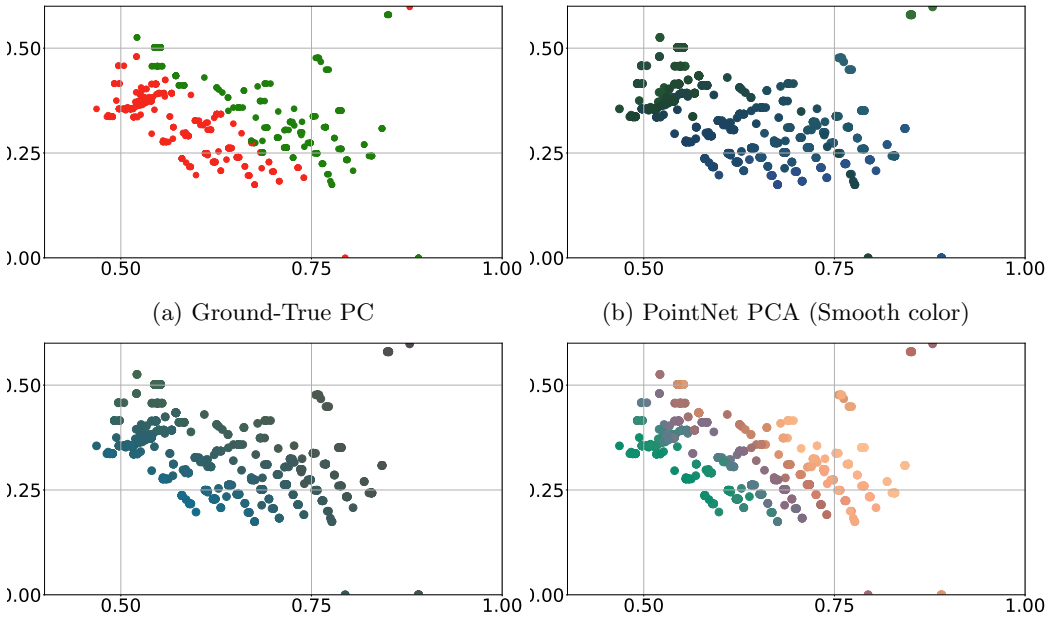
(c) PointNet++ Prediction

(d) DGCNN Prediction

(e) Transformer Prediction

(f) GT Prediction

Figure 5.5.4: Prediction of each model (PCs with $W = 12$, in the robot RF).



(a) Ground-True PC

(b) PointNet PCA (Smooth color)

(c) DGCNN PCA (Smooth color)

(d) GT PCA (Sharp color)

Figure 5.5.5: Zoom-in visualization in a region of interest from Figure 5.5.3

point cloud depicted in Figure 5.5.3 (a) shows that the true and noise points are grouped in separate small communities/structures. Hence, a good representation should have very distinct values (i.e. colours) between clusters of true and noise points. By contrast, a smooth (or completely absent in the worst case) colour gradient reflects a model with poor performance that failed to identify local communities/structures in the PC. Using this understanding and the PCA visualizations from Figure 5.5.3, we now explain the inner workings of each architecture and its advantages and limitations.

PointNet: Figure 5.5.3 (b) shows that the features learned by the PointNet are smooth across space. This is expected since each point feature is learned from the point's 3D coordinates, neglecting the point's neighbourhood. Given its inability to capture the local structures, the PointNet identifies correctly one main region/cluster. The nearby points are smoothly classified as true, while the far points are smoothly classified as false. In most cases, the selected cluster is close to the sensor position ($x = 0, y = 0$), as in the prediction in Figure 5.5.4 (b). This cluster has a higher probability of being true and is high-density (given the sensor's physical properties). As such, its correct classification results in high accuracy but is not necessarily associated with a good global scene understanding.

PointNet++: This architecture, widely used in classical PCs, was designed to address PointNet's inability to capture local-to-global structures through hierarchical learning. The hierarchical learning

should be possible via the sampling of the PC in initials layers and the consequent interpolation in the later layers. However, this has a side effect of over-smoothing, which is too extreme for the particular case of MilliNoise dataset. This is observed first in Figure 5.5.3 (c), in which features from PointNet++ are almost all identical, and then in the prediction (Figure 5.5.4 (c)), where the model misses the proper classification of many true points.

DGCNN: As shown in Figure 5.5.3 (d), the DGCNN is able to detect local clusters within the data. However, the features of the points within these clusters are over-smoothed. The zoom-in in Figure 5.5.5 (c) depicts a cluster with over-smoothed features. This over-smoothing is a result of the DGCNN strategy to rebuild the neighbourhood at each layer using feature similarity. This strategy, combined with the natural smoothing aspect of GNNs, causes the neighbourhoods of each point to remain fixed at each layer. Given a fixed neighbourhood, a point in the DGCNN only explores its local cluster to learn features. As such, the DGCNN is unable to efficiently explore local-to-global geometrical structures within the mmWave PC, leading to a poor performance.

Transformer & ***Graph-Transformer***: In Figure 5.5.3 (e) and (f), we observe the PCA of features from the Transformer and the GT (which combines attention and message-passing). Both have sharp colour variations between points. The sharp variation means that the model is able to learn highly distinct features between points of different classes (noise and true points) in close geometric proximity, an ability the models investigated above do not possess. This ability is achieved by learning attention maps to explore the 3D space. The learned attention grants the network flexibility to explore local-to-global structures for each point. The ability to distinguish local structures is especially noticeable in the cluster close to the sensor, depicted in the zoom-in in Figure 5.5.5. There is a very clear correspondence between the ground-true visualization and GT PCA features, demonstrating GT is able to distinguish between true and false point structures.

The superior results of GT are a direct result of this attention-based strategy to explore the space, which is well-suited for sparse and noisy mmWave PCs. This strategy, combined with message-passing convolution, allows for more effective exploration and understanding of the mmWave dataset's spatial structure.

5.5.5 Ablation Studies

We now turn our investigation to how the input data and the Temporal and Geometric blocks affect the model's denoising performance. In particular, we perform an ablation study on the *(i)* number of frames in the accumulation window; *(ii)* the inclusion of velocity and intensity in input PC; *(iii)* the impact of the *Temporal* block; and (*iv)* the different *Geometric* block implementations.

Table 5.5: Results obtained varying the number W of stacked PC frames (for K-fold-1-6).

Input	Architecture	BCE ↓	Acc ↑	Recall ↑	F1 ↑
Input: 1 frame	PointNet	0.629	0.693	0.599	0.602
	PointNet++	0.680	0.613	0.382	0.434
	DGCNN	**0.591**	0.688	0.512	0.560
	Transformer	0.623	0.666	0.464	0.519
	GT $\setminus o\text{-}Temp$	0.633	**0.716**	**0.651**	**0.640**
Input: 3 frames	PointNet	0.604	0.720	**0.657**	0.646
	PointNet++	0.626	0.654	0.300	0.403
	DGCNN	0.602	0.679	0.459	0.526
	Transformer	0.535	0.730	0.590	0.629
	GT	**0.520**	**0.768**	0.584	**0.662**
Input: 6 frames	PointNet	0.634	0.691	0.504	0.560
	PointNet++	0.687	0.755	0.598	0.655
	DGCNN	0.567	0.716	0.621	0.630
	Transformer	0.525	0.745	0.680	0.675
	GT	**0.502**	**0.782**	**0.711**	**0.717**
Input: 12 frames	PointNet	0.536	0.740	0.623	0.651
	PointNet++	0.619	0.659	0.350	0.445
	DGCNN	0.562	0.712	0.625	0.628
	Transformer	0.499	0.750	**0.725**	0.693
	GT	**0.480**	**0.794**	0.695	**0.725**

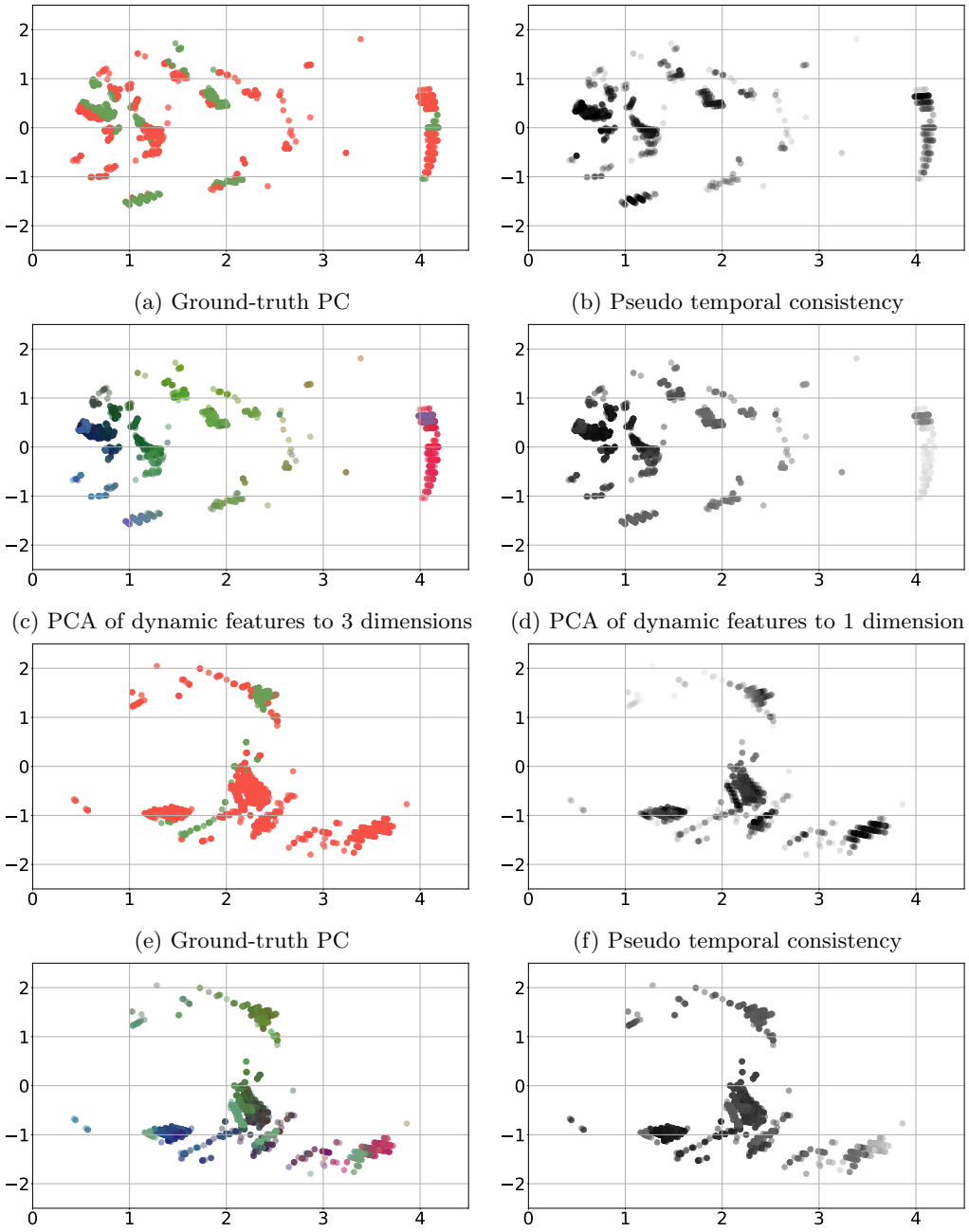
The effect of accumulated point cloud frames

In this subsection, we investigate the effect of PC sparseness on the performance of the considered architectures. Specifically, we test the denoising task with different W values, which represent the number of point cloud frames accumulated in the input point cloud $\mathcal{P}$. Table 5.5 presents the results for $W = \{1, 3, 6, 12\}$, with higher W leading to denser input PCs. The results show that increasing W generally leads to a denoising accuracy improvement, which is expected since increasing W increases the amount of geometrical structures available to the models.

An exception is the PointNet++ model. PointNet++ fails to denoise PCs with $1, 3, 12$ frames, however for $W = 6$ the model achieves an accuracy of 75% and an F1-score of 0.63. In this instance, a point cloud of 6 frames, comprising $1,200$ points, shares the same number of points of the original PointNet++ paper configuration. These results show how fragile and configuration-dependent the PointNet++ is in denoising mmWave PCs.

The effect of including velocity and intensity

We now investigate the effect of various features available from the radar sensor in the denoising task. Table 5.6 reports the model performance when given the additional intensity or velocity in input.



(a) Ground-truth PC (b) Pseudo temporal consistency

(c) PCA of dynamic features to 3 dimensions (d) PCA of dynamic features to 1 dimension

(e) Ground-truth PC (f) Pseudo temporal consistency

(g) PCA of dynamic features to 3 dimensions (h) PCA of dynamic features to 1 dimension

Figure 5.5.6: Comparison of pseudo-temporal consistency (darker colour represents a stronger temporal consistency) with the features learned on *Temporal* block projected to 3 and 1 dimensions. Figure from two PCs examples with $\mathcal{W} = 12$ acquired from the robot reference system.

Table 5.6: The effect of additional variables on the model performance (for K-Fold-1-6) .

Input	Architecture	BCE ↓	Acc ↑	Recall ↑	F1 ↑
xyz	PointNet	0.536	0.740	0.623	0.651
	PointNet++	0.619	0.659	0.350	0.445
	DGCNN	0.562	0.712	0.625	0.628
	Transformer	0.499	0.750	**0.725**	0.693
	GT	**0.480**	**0.794**	0.695	**0.725**
xyz + intensity	PointNet	0.588	0.693	0.578	0.594
	PointNet++	0.610	0.678	0.444	0.517
	DGCNN	0.646	0.622	0.167	0.256
	Transformer	0.563	0.703	0.464	0.548
	GT	**0.450**	**0.796**	**0.726**	**0.734**
xyz + velocity	PointNet	0.564	0.711	0.584	0.610
	PointNet++	0.605	0.678	0.525	0.558
	DGCNN	0.513	0.757	0.671	0.681
	Transformer	0.489	0.759	0.670	0.683
	GT	**0.448**	**0.804**	**0.695**	**0.734**

A small performance improvement is registered by including information on the point velocity in the model input. For example, GT improves accuracy from 79% to 80% and F1-score from 0.72 to 0.73. The intuition is that, due to the nature of the signal noise, real points will share similar velocity values. On the other hand, noise points will have very different values with respect to their neighbour points since they are mainly produced by multi-path reflections. This creates PCs with regions of points with similar (very diverse) values for true (noise) points. To note, the GT architecture was not designed to explore the velocity. The velocity is included in the model input via simple concatenation with the point's coordinates. It is possible that a model specifically designed to exploit velocity could achieve even better performance.

Table 5.6 also shows, contrary to expectations, that the inclusion of intensity as input to the model does not result in improved performance. The reason for the low-performance gain is that, due to the main source of noise being multiple wave reflections, noise points would share a low-intensity value. However, in an indoor environment, this is not the case: the indoor nature of the environment reduces the distance travelled by the waves, resulting in small to no reduction of the intensity of the waves received from noise points and true points. Hence, the addition of such information to the input PC does not improve performance.

The Impact of Temporal Block

In this subsection, we investigate the effect of the *Temporal* block in the denoising task. We begin by evaluating a variation of GT without the *Temporal block*, denoted as GT w\o-*Temp*. The comparison results are reported in Table 5.7 and empirically show that the inclusion of the temporal block leads

Table 5.7: The improvement gained by the Temporal-block (average across the considered train/test folds)

Architecture	BCE ↓	Acc. ↑	Recall ↑	F1 ↑	EMD ↓	CD ↓
GT w*o-Temp*	0.575	0.727	0.660	0.657	0.409	0.208
GT	**0.535**	**0.749**	**0.667**	**0.685**	**0.360**	**0.192**

Table 5.8: Comparison of the *Graph-Transformer* and a *KHop-GNN* (average across the considered train/test folds).

Architecture	BCE ↓	Acc. ↑	Recall ↑	F1 ↑	EMD ↓	CD ↓
KHop-GNN	0.563	0.739	0.629	0.656	0.403	**0.198**
GT	**0.535**	**0.749**	**0.667**	**0.685**	**0.360**	0.192

to significant improvement across all metrics.

To gain a deeper insight into the *Temporal* block, in Figure 5.5.6, we visualize the dynamic features learned by the block and compare them to a *pseudo notion* of temporal consistency. To perform this comparison, we compute a *pseudo-temporal consistency* on the ground truth by averaging, for each point, the distance of the closest points in the immediate past and future frame. Figure 5.5.6 depicts two different point clouds. In particular, the Figure shows from left to right: the ground-true PC; *pseudo-temporal* consistency computed; and the PCA features learned from the *Temporal* block, projected to 3 and 1 dimensions respectively. In Figures 5.5.6 (b) and (f), a darker colour corresponds to a higher consistency of the point in time, while a lighter colour means the point is sporadic. A visual correspondence can be found comparing the calculated pseudo-temporal consistency and PCA features. The darker points in Figure 5.5.6 (b) from the pseudo-temporal consistency correspond to darker blue points in Figure 5.5.6 (c) and (g) (3 dimensions PCA) and to darker points Figure 5.5.6 (d) and (h) (1 dimension PCA). This visual correspondence shows the *Temporal* block is learning how consistent is each point in time, producing more robust and informative features. These features are then passed to the *Geometric* block, which utilizes it to learn attention weights that take into account if points are stable or not.

Geometric Block Design Choice:

In Table 5.8, we experimented with a different architecture design for the *Geometric* block. We investigated if, instead of using learned attention, which can be computationally expensive, we can build neighbourhoods in a *ad-hoc* manner. To this end, we implemented an architecture denoted as *KHop-GNN*. The *KHop-GNN* is a message-passing GNN, where the neighbourhoods increase by a *K-hop* at each layer. For example, at the first layer, the network learns point features to its 1^{st} closest neighbourhoods, whereas at the $l-th$ layer, the network aggregates from the k^{th} closest neighbourhoods.

Table 5.8 shows it was possible to approximate the performance of the GT with a *KHop-GNN*, which achieved $\sim 74\%$ accuracy. However, it is important to note that the design *Khop-GNN* has to be manually tuned. The neighbourhood (the jump between layers) has to be manually pre-defined and is dependent on the size of the input point cloud, whereas the GT can handle PC of any size.

5.5.6 Computational Complexity

We evaluated the computational complexity of the proposed GT model and found it comparable to state-of-the-art architectures. The GT contains 1.6M trainable parameters, which in the context of PC processing is considered a moderate-size model. It can process the entire training dataset in 243 seconds on single Tesla V100S-PCIE GPU with 32 GB of dedicated memory. These results mean the GT model can be considered for practical real-world applications.

5.5.7 Limitations and Future Work

We believe our method can be improved in several ways. Firstly, the denoising performance of the GT can be further improved. While it is a good early approach, with a significant improvement over the state-of-the-art, Figures 5.5.1, 5.5.2, and 5.5.4 show our proposed model still misclassified a considerable amount of points. Secondly, the metrics to evaluate the performance of the denoising models have a limited ability to measure how well the scene is understood. We have partly addressed this challenge by proposing a metric that evaluates the geometry of the denoised point cloud (EMD and CD) instead of per-point accuracy. However, such metrics are still point averages, with every point having the same impact on the final score, and more representative metrics can be designed. Lastly, we did not take advantage of the distance to the closest object associated with each point provided by MilliNoise dataset. We believe this metric can be incorporated into the loss function to train more sophisticated and robust models. We leave the above directions as future works.

5.5.8 Concluding Remarks

This chapter focused on point-wise denoising of mmWave PCs, where the goal is to classify every point either as true or false. To this end, we leveraged MilliNoise [104], the dataset introduced in Chapter 3.6. A benchmark of the performance of the most common point cloud processing approaches is provided, showing their limitations in exploring the local-to-global structures in sparse and noisy point clouds. To address the identified limitations, a graph-based transformer architecture denoted as GT is proposed. The GT leverages both temporal and geometric cues for accurate denoising, achieving 75% accuracy, corresponding to 5% gain over the state-of-the-art. The superior

performance of GT-MilliNoise was confirmed in visual results, where the denoised PCs appeared clearer with sharper object boundaries.

This chapter is to be intended has a first step towards mmWave PCs denoising. In fact, this chapter highlighted a gap in the literature and has shown the limitations of state-of-art architectures for such peculiar PCs. An interesting future step is to bring the proposed architecture to a smaller size, in order to be able to deploy it on embedded devices, making it an effective resource in the mmWave radar context.

Chapter 6

APEIRON: a Multimodal Drone Dataset Bridging Perception and Network Data in Outdoor Environments

6.1 Introduction

Unmanned Aerial Vehicles (UAVs), also known as drones, have grown popular in numerous domains, including surveillance [105], disaster response [106], environmental monitoring [107, 108], live drone broadcasting [109–111], and Internet-of-Drones (IoD) [112]. In these context, a fundamental requirement for such vehicles is full autonomy, i.e. the capability of fulfilling complex missions in Beyond Visual Line-Of-Sight (BVLOS) scenarios. For this purpose, drones need precise environmental perception to navigate their surroundings efficiently. Various sensing technologies can be utilized, including Time-of-Flight (ToF) sensors (e.g., LiDARs), radars, RGB cameras, depth cameras, and more. Simultaneously, drones often rely on long-range wireless communication systems, such as cellular networks, to transmit real-time data, including telemetry, sensor outputs, or live video feeds, to ground operators or live audiences [113]. This communication layer plays a crucial role in enhancing situational awareness in applications like disaster response and surveillance. Additionally, it is essential for coordinating drone swarms [114,115] and supporting various entertainment scenarios, such as live streaming sports events or providing news coverage [116].

In this context, as learning-based algorithms are increasingly employed to develop efficient end-to-end autonomous drone systems, the availability of extensive public datasets capturing real-world environments becomes essential. Existing literature offers various datasets collected using UAVs, which can generally be categorized into two types: 1) *perception datasets* [117–119], focused on data

for tasks such as computer vision and visual SLAM, and 2) *network-related datasets*, centered on scenarios like the Internet of Drones (IoD) [113,120]. However, to the best of the authors' knowledge, no publicly available dataset combines data from perception sensors with network measurements.

This chapter presents APEIRON [121], a comprehensive multimodal aerial dataset that combines perception data from multiple sensors with network bandwidth measurements obtained via an LTE module. Specifically, for perception, a custom-built hexarotor drone is equipped with: 1) a stereocamera (the ZED2i by Stereolabs[1]) and 2) an Event-Based camera (the PropheseeEVK4HD[2]). Additionally, the PixHawk Flight Control Unit (FCU) onboard the drone provides data from GPS, gyroscope, accelerometer, magnetometer, barometer, and environmental temperature sensors.

It is worth noting that while RGB-D cameras offer valuable information about the scene, including point clouds and depth images, their performance can be significantly affected by sudden changes in lighting conditions, such as transitioning from a dark room to a brightly lit area. These limitations, due to the camera's restricted dynamic range, can cause glare effects when the camera faces a light source (e.g., the sun), often resulting in inaccurate or erroneous environmental perception.

Event-based cameras [122] are light-sensitive sensors designed to overcome the limitations of traditional RGB cameras. Unlike conventional cameras that measure absolute light levels, these devices consist of an array of pixels that independently detect changes in light intensity. Each detected change, referred to as an *event*, is generated asynchronously, allowing these sensors to operate at frame rates that are several orders of magnitude higher than those of standard RGB cameras.

On the network side, an LTE module is employed to perform active TCP measurements of both downlink and uplink. The collected network data includes metrics such as estimated available bandwidth, as well as low-level indicators like RTT, congestion window size, and packet loss rates.

The resulting dataset integrates both perception and network data, supporting not only traditional applications in perception or communication but also cross-domain use cases that combine these two types of information. The APEIRON dataset is expected to promote interdisciplinary research at the intersection of multimedia systems, computer networks, and robotics.

6.2 Related Work

This section examines datasets closely related to the one introduced in this work. Specifically, publicly available datasets are categorized into two groups: 1) datasets focused on network-related data collected from drones, discussed in Section 6.2.1, and 2) perception datasets gathered using

[1] `https://store.stereolabs.com/en-it/products/zed-2i`
[2] `https://www.prophesee.ai/event-camera-evk4/`

drones equipped with various sensors, covered in Section 6.2.2.

6.2.1 Network-related datasets

In [123], a study investigates the performance of Long Term Evolution (LTE) networks in a rural environment using drones. The focus is on the interference caused by drones and its impact on the utilization of network resources.

The work in [113] presents raw LTE data collected by drones operating in both rural and urban-like environments. This dataset is designed to improve UAV communication and navigation while analyzing cellular coverage at different altitudes. The paper also includes post-processing tools and experimental code to support further research.

Similarly, [120] offers an analysis of data gathered from multiple drone flights at various altitudes and locations, using different cellular carriers in a medium-sized urban area. The dataset, along with details about the operating system, chipset, drone instrumentation, and server-side packet captures, is made available as an open-source resource for the research community.

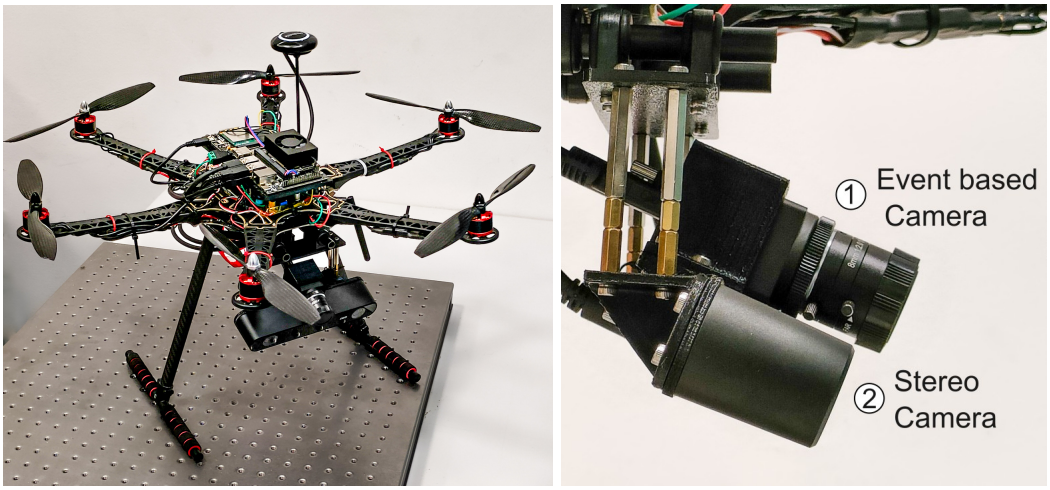
While these datasets focus on network-related data, they do not include any perception data.

6.2.2 Perception datasets

In [124], the authors propose a method for tracking power lines using a drone equipped with an event camera similar to the one employed in this work. The approach identifies power lines in the event stream by detecting planes in the spatio-temporal signal and tracking them over time. Evaluated in real-world flights along a power line, the method demonstrates significant improvements over existing techniques. While this study highlights the effectiveness of event cameras for industrial applications, the associated dataset is not multimodal, as it includes only perception data from the event-based camera and lacks any network-related data.

The work in [125] introduces a dataset for high-speed drone racing. It includes over 27 sequences captured in indoor and outdoor environments, covering more than 10 km of flight distance using a first-person-view (FPV) racing quadrotor operated by an expert pilot. The dataset features camera images, inertial measurements, event-camera data, and precise ground truth poses. Although it provides valuable insights into FPV quadrotors, it focuses on a specific application and does not include any networking data.

In [118], the authors present DroneFace, a dataset of facial RGB images captured by a camera mounted on a long stick, simulating images taken by a drone at various distances and heights. The dataset includes frontal and side portrait images of subjects, aimed at facial recognition training. Similarly, [117] offers an aerial dataset captured at altitudes ranging from 50 to 500 meters. It



(a) Hexarotor drone (b) Sensors placement

Figure 6.3.1: Hexarotor drone and sensors placement

consists of monocular RGB images of a roundabout in Cyprus at a resolution of 3840×2160. The dataset is annotated to study how drone altitude affects the detection of cars.

The dataset most closely related to ours is discussed in [119]. Specifically, it includes data from i) a mmWave radar sensor, ii) a full HD RGB camera (1080p), iii) a 240×180 resolution event-based camera, and iv) a gyroscope. However, there are key differences between the dataset presented in this paper and that of [119]. First, [119] focuses on collision avoidance tasks in indoor environments, while our dataset is collected in outdoor scenarios, offering a broader range of potential applications. Additionally, our dataset utilizes a higher-resolution event-based camera (1080×720, compared to 240×180) and includes RGB-D images instead of RGB images. The inclusion of depth information from the RGB-D data provides ground truth, enabling applications such as monocular depth estimation from the event-based camera. Finally, unlike [119], the APEIRON dataset proposed in this work incorporates network measurements.

6.3 Methodology

This section outlines the methodology used to collect the APEIRON dataset. Section 6.3.1 details the data collection platform, including information about the drone and onboard equipment utilized. Following this, Section 6.3.1.1 describes the software stack developed to facilitate data collection.

Table 6.1: Sensors and hardware employed for the collection of perception and networking data

Device	Description
Prophesee EVK4 HD	*Data type*: Events *Sensor*: Sony IMX636 HD 1280×720 pixels, 1/2.5" CMOS *Pixel latency*: ¡ 100 μs *Dynamic range*: $\geq$ 120 dB
Stereolabs ZED2i	*Data types*:RGB-D (L/R monocular and depth), point cloud, acceleration, angular rate, magnetometer, barometric pressure, temperature, pose estimate *Sensor*: 2688×1520 pixels, 1/3" CMOS, 2.1 mm focal length *FoV*: Horizontal 110°, Vertical 70° *Baseline*: 12 cm *Depth range*: 20 m *Capture res.*: 720p@60 fps *Dynamic range*: 64.6 dB *Positional tracking accuracy*: $\pm$1 mm *Orientation accuracy*: $\pm$0.1°
PixHawk 2.4.8 (FCU)	*Data types*: IMU data, GPS data, barometric pressure, temperature, flight logs
Quectel EG25-G LTE module	*Max Bandwidth*: 150Mbps DL/ 50Mbps UL

6.3.1 Data collection platform

Figure 6.3.1(a) displays the self-built hexarotor drone used to collect the APEIRON dataset. The sensors and hardware essential to the dataset's features are summarized in Table 6.1.

In the following, more details on the employed drone, sensors, and their placement on the drone are provided.

UAV frame and propulsion

The hexacopter is constructed using the DJI S550 frame, selected for its ample space and payload capacity. The propulsion system consists of six 2212 920,KV brushless motors paired with 1045 propellers, powered by 30,A ESCs and a 4S battery. This configuration supports a total weight of 3,kg, including the drone, battery, sensors, and computing equipment.

Flight controller unit

The FCU used is the off-the-shelf *Pixhawk 2.4.8*, configured with the latest stable PX4 version[3]. This unit is responsible for stabilizing and controlling the UAV's position, enabling the execution of autonomous missions defined by waypoint lists. In this work, it is utilized in two primary GPS-aided flight modes: 1) *Position flight mode*, where the UAV is manually operated to allow precise control of the path when inspecting nearby targets or for testing purposes; 2) *Mission flight mode*, where a set of waypoints and procedures is loaded onto the FCU for repeatable and autonomous operation.

Perception sensors

Stereocamera: Stereolabs' ZED2i is an RGB-D camera equipped with an internal accelerometer, gyroscope, magnetometer, barometer, and temperature sensor. It is supported by a Software Development Kit (SDK) that facilitates the computation of depth images, self-pose tracking, and environment reconstruction in the form of point clouds.
Event-Based camera: The EVK4 HD is the latest evaluation kit produced by Prophesee, featuring the IMX636, a high-definition CMOS sensor with a pixel latency of $100, \mu s$ and a dynamic range exceeding 120,dB. Event processing is embedded on the device, enabling very fast event detection and processing. The sensor achieves a temporal resolution greater than 10,000 FPS. Data from this sensor are stored in a raw format, consisting of events encoded in the EVT3 format. Each detected event includes: 1) the X and Y *position*, 2) the *event polarity*, a boolean indicating whether brightness has increased or decreased, and 3) the event *timestamp*[4]. These data can be exported in post-production to various formats (e.g., CSV) for easier utilization.
Sensors displacement: The FCU, along with its onboard sensors, is mounted on the main plate of the drone at its center (zero displacement). The sensors are arranged to be closely aligned and share the same field of view, as illustrated in Figure 6.3.1(b). Custom mounting plates were designed and 3D printed to securely attach the event-based camera to the stereo camera, which already provided mounting points. Additionally, a separate mounting plate was designed to attach the stereo camera to the drone. This mounting plate is angled at 30° with respect to the plane on which the FCU is positioned. This configuration ensures that the sensors have a field of view that can capture both ground and forward perspectives, enabling features useful for various applications such as environment reconstruction, self-localization, and more.

[3]At the time of writing, version 1.14.0
[4]Expressed in μs relative to camera activation

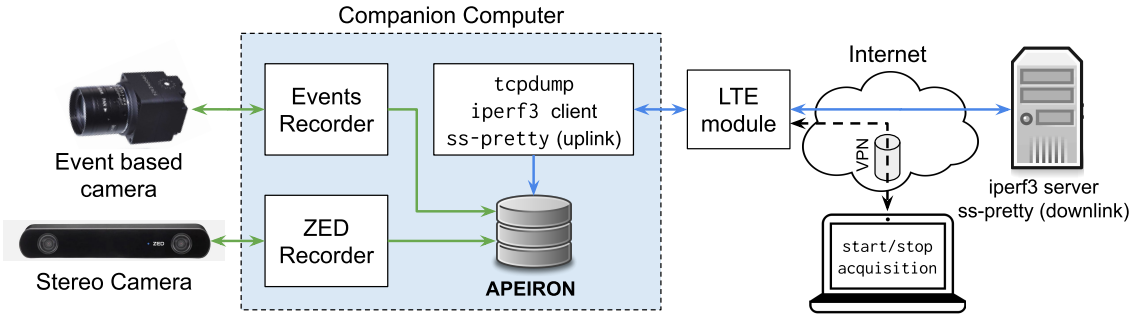


Figure 6.3.2: Software and connection architecture composed by: the Jetson Xavier on-board of the UAV, the remote server used as target for the uplink/downlink network traces, the user SSH interface used to start/stop and monitor the collection process via a remote shell.

Computing

The companion computer is an *NVIDIA Jetson Xavier NX 16 GB* equipped with a 512 GB SSD storage device. This board is utilized to collect sensor data and perform network measurements. The software stack developed to gather sensor data is outlined in Section 6.3.1.1.

Connectivity

Communication with the FCU is facilitated by a remote radio controller and a Holybro SiK telemetry module, while the connectivity of the companion computer is enabled by the *Quectel EG25-G* LTE module[5].

6.3.1.1 Software stack

Figure 6.3.2 shows the software architecture used for the collection of perception and network data.

Data collection necessitates synchronization across all data sources to enable the use of multi-modal datasets. To achieve this, we record timestamps for each data source, ensuring synchronization during post-processing. A Docker image has been developed for each data source collection. Each running Docker container writes data to a dedicated directory, mounted as a volume. The data collection process is orchestrated using the Docker *compose* tool.

Perception

Stereo camera. Regarding the ZED2i camera, we utilized a Docker image[6] provided by StereoLabs to record live feed information in a proprietary format known as SVO, using the *ZED_SVO_Recording*

[5] `https://www.quectel.com/product/lte-eg25-g`
[6] stereolabs/zed:4.0-tools-devel-l4t-r35.2

tool[7] from the ZED SDK 4.0. This tool simulates a real camera by reading recorded SVO files, enabling the playback of captured data, including video feeds and sensor information. This functionality supports offline use of the standard SDK, allowing the generation of depth maps, dense point clouds, pose estimates, and mesh maps using higher computation resources on a workstation, which are not feasible with embedded systems like the Jetson Xavier. The SVO format also includes UNIX timestamps for each frame and sensor data, facilitating synchronization with other data sources during post-processing.

Event camera. Regarding the EVK4 HD, the OpenEB[8] SDK, an open version of the Metavision SDK provided by Prophesee, was employed to record "raw" files storing EVT3 data and "bias" files representing the calibration parameters used for the camera. This enables the playback and manipulation of the entire event camera data flow. Events are stored alongside timestamps, expressed as the number of microseconds elapsed since the capture started. To ensure accurate time synchronization, a text file containing the UNIX timestamp at which the event-based camera was activated is also provided. In post-production, events can be exported to CSV files along with their timestamps, simplifying the use of event data without requiring direct interaction with the Metavision SDK.

Network

Network traces are obtained by initiating a TCP flow in either the downlink or uplink direction using the `iperf3` tool. Specifically, an `iperf3` server instance is run on the remote server, while the client runs on the onboard computer. Two types of data are collected: 1) TCP traffic packets stored in PCAP format using `tcpdump`, and 2) timestamped socket statistics such as RTT, Goodput, congestion window size (cwnd), slow start threshold (ssthresh), and packet losses, collected by adapting `ss-pretty`[9] to our requirements. The process is similar for both uplink and downlink measurements, with the only difference being the location of `ss-pretty`. Since the congestion control algorithm operates at the sender, `ss-pretty` is run either on the onboard computer to measure uplink traffic or on the remote server for downlink traffic.

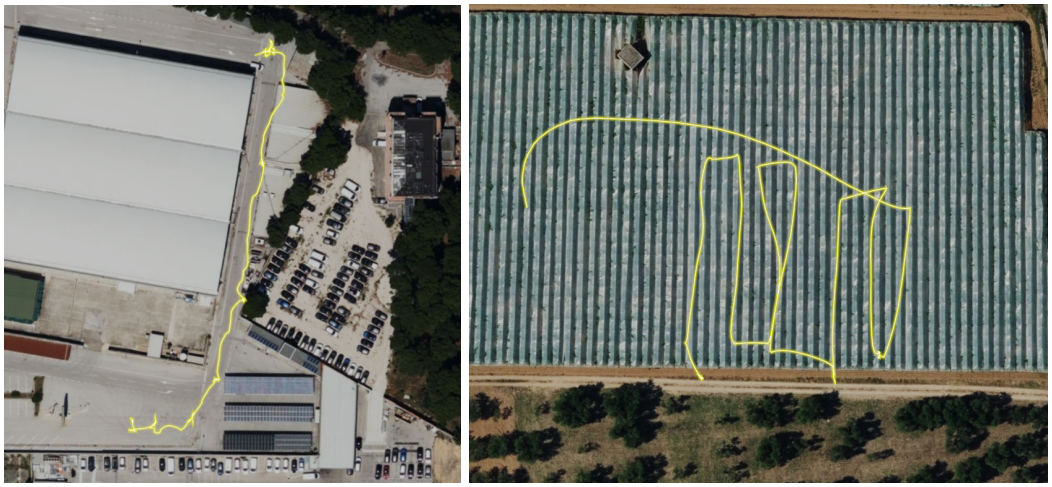
Table 6.2: APEIRON dataset main features

Run	Flight duration [min]	RGB frames	Avg events rate [Mev/s]	EVT3 vs SVO File size [GB]	GoodPut [Mbps]	Average RTT [ms]	Direction	PX4 Online Flight Preview
IND-1	03:20	10324	3.6	2.5 vs 7.1	2.389	184.271	Upstream	http://bit.ly/49kb3LE
IND-2	04:02	12031	3.6	3.0 vs 8.3	N/A	N/A	Downstream	https://bit.ly/48XWCNt
IND-3	04:11	11299	5.5	4.4 vs 7.9	4.372	190.886	Downstream	https://bit.ly/490x1DE
IND-4	03:58	11079	4.7	3.7 vs 7.7	N/A	N/A	Upstream	https://bit.ly/48YQB3a
OF-1	04:14	11142	6.0	4.6 vs 9.9	9.721	412.106	Upstream	https://bit.ly/42y38YY
OF-2	02:15	5520	7.5	2.9 vs 5.0	3.873	144.713	Downstream	https://bit.ly/3HMtMni
OF-3	03:07	9220	9.2	4.8 vs 7.7	7.003	493.119	Downstream	https://bit.ly/3SItrs6
OF-4	03:35	9513	11.8	7.6 vs 9.7	6.477	104.132	Upstream	N/A

6.4 The APEIRON dataset

In this Section, the APEIRON dataset is presented. Section 6.4.1 provides a description of the included scenarios, then, Section 6.4.2 presents the dataset format and, finally, Section 6.4.3 analyses the dataset from a quantitative point of view.

6.4.1 Scenarios



(a) Industrial Scenario (b) Open Field Scenario

Figure 6.4.1: Example scenarios and trajectories

Currently, the APEIRON dataset captures two distinct scenarios: an industrial scenario (IND) and an open field scenario (OF). Multiple trajectories were conducted at varying heights within each scenario to capture diverse perspectives. These scenarios were selected to highlight the limitations and strengths of the sensors employed.

[7] https://github.com/stereolabs/zed-sdk/tree/master/recording
[8] https://github.com/prophesee-ai/openeb
[9] https://github.com/gaddman/ss-pretty

In particular, the first scenario, illustrated in Figure 6.4.1(a), is located near an *industrial building*, where the drone follows various paths to capture the sides and surroundings of the building. This scenario can be useful, for example, in the study of Visual Inertial Odometry (VIO) techniques [126] or 3D reconstruction applications. The scenario alternates between frames rich in visual features and frames with very few detectable features, posing distinct challenges for perception and sensor performance.

Figure 6.4.1(b) illustrates the *open field* scenario, where the drone captures data in an expansive open area at varying altitudes. In this scenario, the lack of visual features poses significant challenges for Visual Inertial Odometry (VIO) applications, as the sparse environment reduces the number of identifiable landmarks.

To conduct these trajectories, two approaches were employed: i) manually controlled flights in position mode, used for close inspections of buildings to ensure safe operation; and ii) autonomous flights, guided by the *QGroundControl* tool, which defines waypoint lists for the FCU, allowing for smoother, repeatable trajectories.

6.4.2 Dataset format

Data collected in APEIRON are organized in *runs*: trajectories executed by the drone in a given scenario and identified by the timestamp, expressed as "run-*day-month-year-hours-minutes-seconds*". Each run is associated to a directory, containing:

- `event.bias`: a file describing the biases employed to capture data from the Event-Based camera;
- `event.raw.timestamp`: a file containing the initial UNIX timestamp used as offset for the raw events, relative to the initial boot of the camera;
- `event.raw`: the file collecting the events, produced by the *Metavision* SDK;
- `zed2i.svo`: the recordings of the *ZED2i* camera, obtained from the *Stereolabs*' SDK;
- `log-`*`day-month-year-hours-minutes-seconds`*`.ulg`: it contains the flight logs, containing all the data captured by PixHawk's sensors, such as GPS data, accelerometer, magnetometer, control inputs for the motors, etc;
- `tcpdump-down(up).pcap`: the TCP packets capture for the down(up)-link communication to a remote server, collected using *tcpdump*;

- `tcp-internals-down(up).log`: output of the *ss-pretty* tool, leveraged to capture the socket related statistics;
- `run.json`: it contains metadata of the run.

The dataset's Github repository also provide preprocessing tools and Jupyter notebooks which implement the starting functionalities to access data.

6.4.3 Analysis of the dataset

The main features of APEIRON are summarized in Table 6.2. In particular, the dataset collects a total of 100 GB of data for an approximate 30-minute flight time.

To facilitate the review of flight data, Table 6.2 includes links to the *PX4 Flight Review* online tool for each flight run. Additionally, the dataset features an uncontrolled crash (OF-4). We opted to retain such data to offer researchers valuable insights into failure events, enabling the development of techniques such as early detection and mitigation strategies.

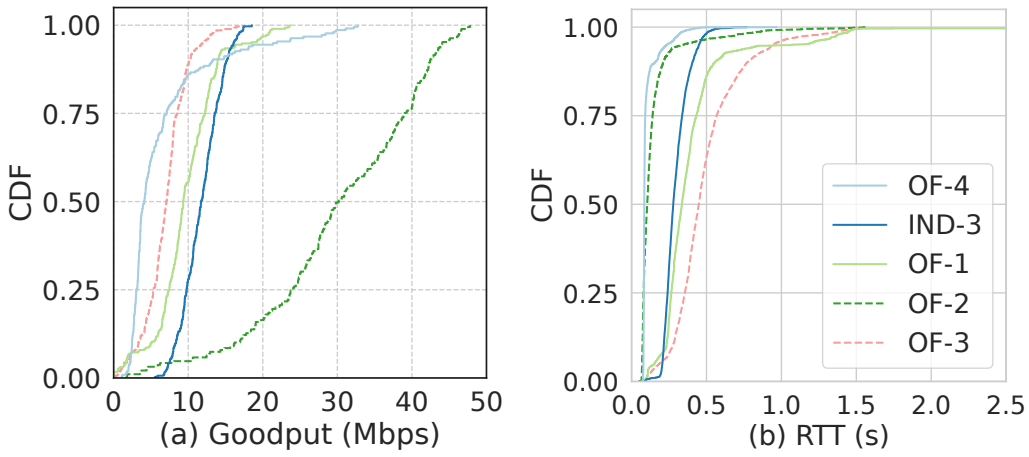
Network traces

To provide insights into the collected network traces, Figure 6.4.2(a) and (b) depict the CDFs of goodput and RTT for each run, respectively. Regarding the goodput (Figure 6.4.2(a)), the highest median value is observed in the open field scenario (OF-2), particularly in the downlink direction, as expected. However, this scenario also exhibits a higher standard deviation compared to the other runs.

The RTT distributions shown in Figure 6.4.2(b) reveal one case with a heavy-tailed distribution (e.g., OF-1). Upon inspecting the logs, these outliers (RTT ≈ 7 s) were identified toward the end of the run. The remaining runs exhibit RTT distributions typical of mobile scenarios. In industrial scenarios, while the median RTT is lower than in the open field scenario, the standard deviation is also reduced.

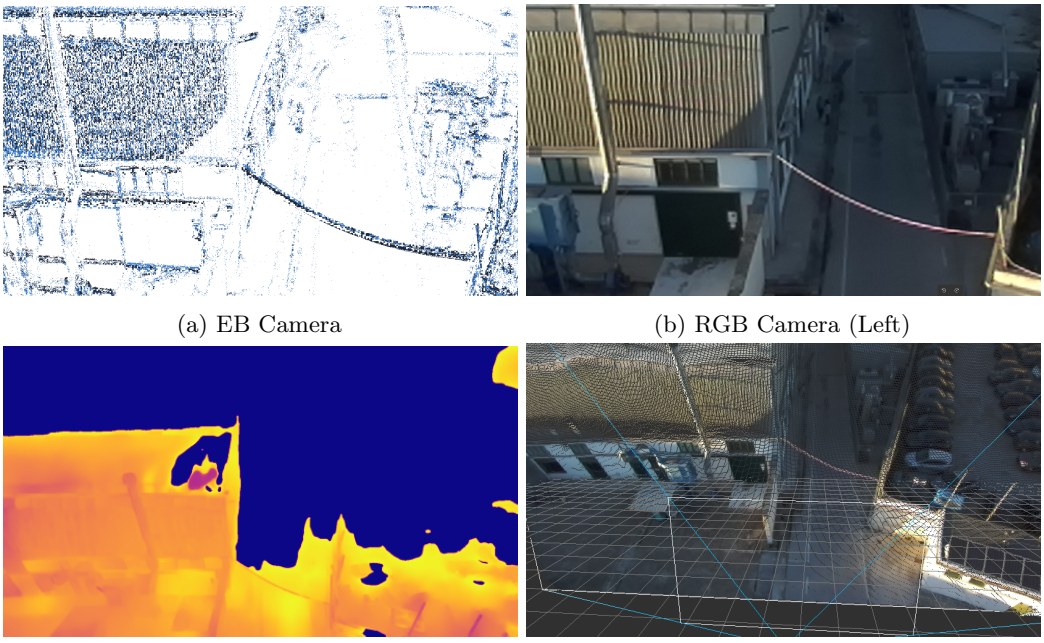
6.4.4 Perception

To offer a clearer representation of the sensor data, Figure 6.4.3 displays frame samples from the IND-4 run for various sensing modalities. Specifically, the figure illustrates a frame from each of the following: the EB camera, the RGB image, the depth image, and the corresponding point cloud from the IND-4 run. To ensure a fair comparison between RGB and EB images, the RGB image has been appropriately cropped to match the field of view (FOV) of the EB camera.



(a) (b)

Figure 6.4.2: Network traces: continuous (dashed) lines refer to uplink (downlink) measurements.



(a) EB Camera (b) RGB Camera (Left)

(c) Depth Camera (d) ZED Local Point Cloud

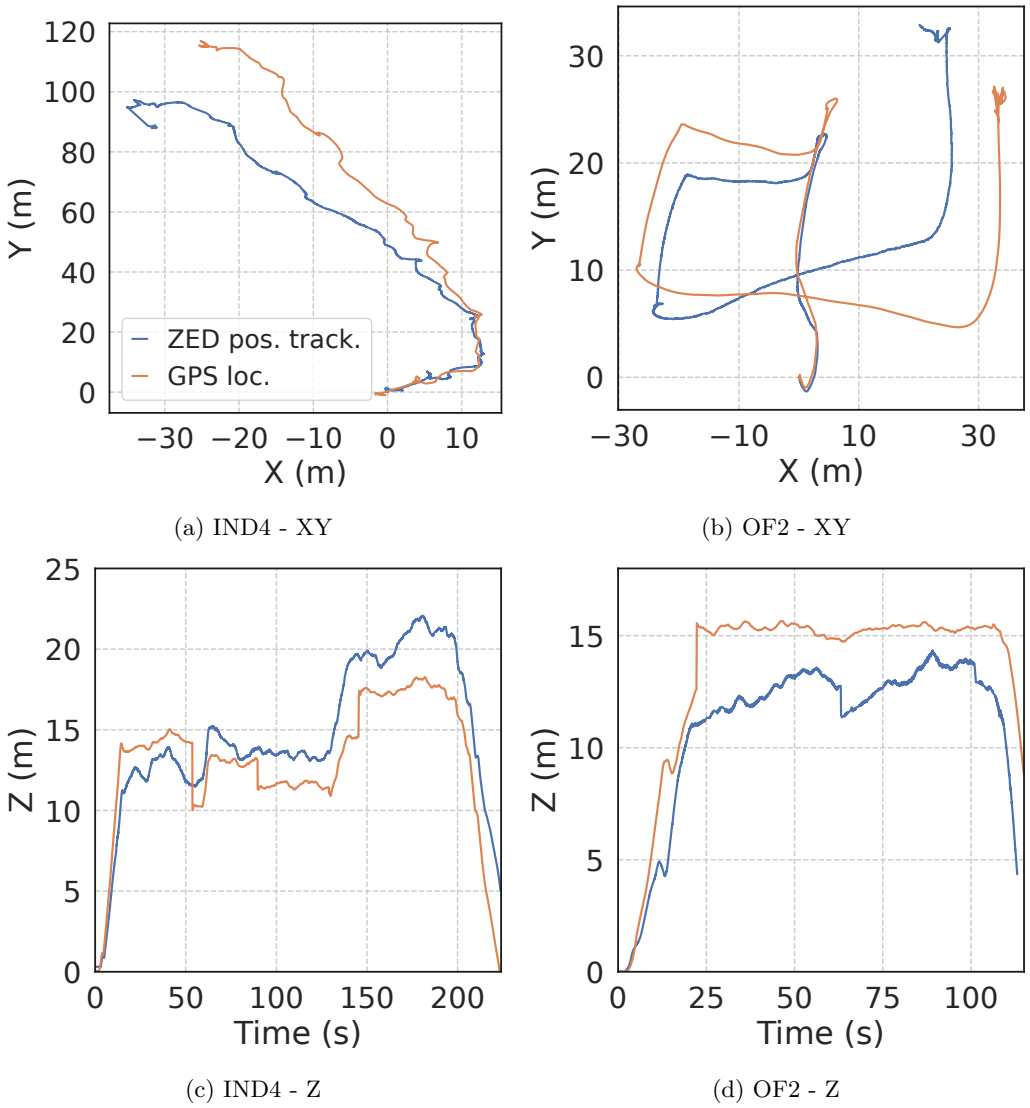
Figure 6.4.3: Perception data (run IND-4)

An interesting comparison arises when examining the GPS position against the pose estimate generated by the ZED SDK's Positional Tracking tool, which utilizes data from the SVO file to compute the estimate. Figure 6.4.4 compares the trajectories obtained from these two localization systems across two distinct scenarios. The first scenario is industrial (IND4, Figures 6.4.4(a) and 6.4.4(c)), characterized by a rich abundance of visual features. The second scenario is an open field (OF2, Figures 6.4.4(b) and 6.4.4(d)), featuring a significant lack of visual features. The figure examines trajectory comparisons along the X and Y axes (Figures 6.4.4(a) and 6.4.4(b)) and along the Z axis by flight duration (Figures 6.4.4(c) and 6.4.4(d)). As anticipated, the ZED SDK's Positional Tracking shows noticeable drift in both scenarios, with a more pronounced impact in the open field due to the scarcity of useful visual features.

6.5 Concluding Remarks

This chapter introduces APEIRON, a multi-modal public dataset combining perception and network data collected from drones in outdoor scenarios. The dataset incorporates various types of sensors, including a high-resolution event-based camera, an RGB-D camera, gyroscope, accelerometer, and GPS data, offering rich and diverse information for a wide range of applications. APEIRON stands out due to its unique combination of networking and perception data in outdoor environments, making it a valuable resource for multidisciplinary research at the intersection of multimedia systems, computer networks, and robotics.

The dataset presented here holds significant potential for applications such as object detection, tracking, and recognition, as well as multimedia systems involving real-time streaming of sensor data to a Ground Control Station (GCS). APEIRON is designed to facilitate further research and innovation in the fields of perception and networking for drones.



(a) IND4 - XY

(b) OF2 - XY

(c) IND4 - Z

(d) OF2 - Z

Figure 6.4.4: Localization using GPS or ZED Positional Tracking

Part II

Control

Chapter 7

Sample Efficient Reinforcement Learning for pose regulation of mobile robot

7.1 Introduction

Learning-based model-free approaches are rapidly gaining interest in the field of robotics due to their ability of controlling robots without requiring the knowledge of their model, while fully leveraging their maneuverability and dexterity. Such approaches allow to design robots that learn from the environment and are able to adapt to successfully carry out new tasks. Reinforcement Learning (RL) is an unsupervised machine learning approach that seeks to produce an optimal *policy*, with respect to the goal it has to accomplish, by *training* an agent over the collected experience, typically obtained in a trial and error process.

In the literature, the RL algorithms commonly employed in control applications with continuous controlled variables are the *Proximal Policy Optimization* (PPO) [25] and the *Deep Deterministic Policy Gradient* (DDPG) [38]. Another promising technique recently proposed is the *Self Supervised Reinforcement Learning* (SSRL) [127], which leverages an imitation learning approach with a self-supervised replay buffer.

The size of the collected experience required for convergence to an optimal policy is related to as the *sample efficiency* of a RL algorithm. In industrial scenarios, collecting experience, i.e., interacting with the environment, might be costly, especially in the initial phases of the training where the safety of the surroundings and equipment involved might not be granted. Hence, designing a sample-efficient RL algorithm is a key issue to decrease both costs and chances of damage.

An interesting approach recently proposed in the literature to tackle such an issue is *Curriculum Learning* (CL) [128]. The main finding of [128] is that the key to improve RL agents' sample

efficiency and generalization capabilities is the fashion through which examples (or task goals) are proposed to the agent.

In this chapter, a methodology is proposed to efficiently train a RL agent to equip with pose regulation functionalities a four steerable and driving wheels mobile robot. Notice that the trained policy needs to drive concurrently a total of 8 motors. In the literature, classical control techniques have been proposed to control this type of platforms. Some approaches impose added constraints on the wheels [129, 130], others leverage a different type of steering, thus simplifying the kinematics of the robot and limiting its maneuverability [131]. Instead, in this chapter an RL agent is trained without imposing any constraints on the actions, thus allowing the agent to freely impose independent actions on the 8 motors.

To this end, two tools are provided to control such a mobile robot, namely (i) a *Difficulty Manager (DM)* and (ii) an *Episodic Noise*, specifically designed to improve the sample efficiency, exploration effectiveness, and robustness of a RL algorithm [132].

Rather than partitioning the space into zones to present goals with increased difficulty as proposed in [133], the proposed DM samples goals from a negatively skewed distribution whose average is shifted further only when an improvement in the agent's performance is measured. The resulting agent efficiently learns a subset of effective actions already in the first few epochs, exhibiting improved exploration, and next improves performance by refining the learned actions. The trained agent can fully exploit the capabilities of the mobile robot, without introducing limitations and/or constraints on the actions as typically performed in the literature.

7.2 Background

The literature review of this chapter is organized as follows: in Section 7.2.1 a brief introduction to the classical control approaches is proposed, underlining the main techniques and possible criticalities. In Section 7.2.2 a brief introduction to RL is given.

7.2.1 Classical Control of a 4WS4WD

Usually, when approaching the problem of controlling a robot, one can study the dynamical properties of the robot in order to derive a kinematic model of it. With this representation, one can derive information about odometry (*forward kinematic*), or can compute the actions on the actuators necessary, f.i., for path tracking tasks (*inverse kinematic*). However, as the number of actuators and the complexity of the dynamics grow, it becomes more and more difficult and time consuming to obtain such kinematic model. When the kinematics are particularly complex, another solution in

order to derive the model is to introduce some constraints [129], [130], or simplifications that can accelerate the process of deriving said model. Another classical approach is to represent the robot's kinematic model through a different representation, f.i., its Instantaneous Center of Rotation (ICR). The ICR can be expressed in Cartesian coordinates [134], or polar coordinates [135]. However, in this representation there exist singularities in which the steer angle cannot be defined: this is due to the ICR passing by (or near by) the steer axis. This problematic can be tackled through representing these regions as obstacles to be avoided [135], [136] by constructing some potential fields in the steer axis. However, these solutions introduce some limitations in the dynamics of the robot, hence limiting its maneuverability and reducing the set of linear and angular velocities the robot could achieve. In [137], a kinematic model that solves the singularities through a damping effect on the wheels is proposed.

However, in real case scenarios, the same industry may want to employ a different robot over time, or replace the old one, or, simply there might be a request to pursue the same task exploiting a different robot. With a classical approach this would mean to start by scratch.

7.2.2 RL-based Control of a 4WS4WD

A learning-based approach can overcome these difficulties, since the agent, as it explores the action space, can find configurations that leverage the capabilities without neither introducing constraints and/or simplifications, or relying on any robot's model, making it simpler to adapt to different use cases.

There exist many approaches in literature that suite these problems. One of them is the *Deep Deterministic Policy Gradient* (DDPG) [38], an actor-critic method which deterministically produces continuous actions through its actor and updates the parameters through the gradient

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\ p_\pi}[\nabla_a Q^\pi(s,a)|_{a=\pi(s)} \nabla_\theta \pi_\theta(s)] \tag{7.1}$$

where $Q^\pi(s,a) = \mathbb{E}_{s\ p_\pi, a\ \pi}[R_t|s,a]$ represents the critic, that approximates the expected return.

In [138] a set of improvements have been proposed to the DDPG algorithm, resulting in the so-called *Twin Delayed DDPG* (TD3). The solutions involve an added critic, used to increase the stability on the approximation of the expected return, a delay over the actor's update, with respect to the critic's update rate, and the introduction of a Gaussian noise over the actions sampled from the target actor during training.

Another interesting RL algorithm is represented by the *Proximal Policy Optimization* (PPO) algorithm, presented in [25]. This algorithm updates the agent following the policy gradient fashion

and leverages an actor-critic structure, however, in this case, the critic approximates the value function $V(s_t)$. The parameters' update is guided by the *advantage function* $A_t = R_t - V(s_t)$, with $V(s_t) = \mathbb{E}[R_t|\pi_\theta, s_t]$. Another key difference of PPO compared to DDPG sits in the process of action sampling: while in DDPG the outcome of the policy approximator directly represents the actions, in PPO the outcome of the approximator represents the average and variance of the distribution over each action. Hence, for each action two outcomes are returned and a sampling step is required. If on one hand, this is particularly helpful in calculating the log-probability of each action, on the other hand it doubles the number of predictions for each action in the action space.

A recent RL algorithm which has drawn our attention is the so called *Self Supervised Reinforcement Learning* (SSRL), presented in [127]. This RL algorithm trains a policy in a supervised fashion, by labelling each transition with the episode's reward. The replay buffer only stores the first n transitions ordered by the episode's return value, n being the dimension of the replay buffer.

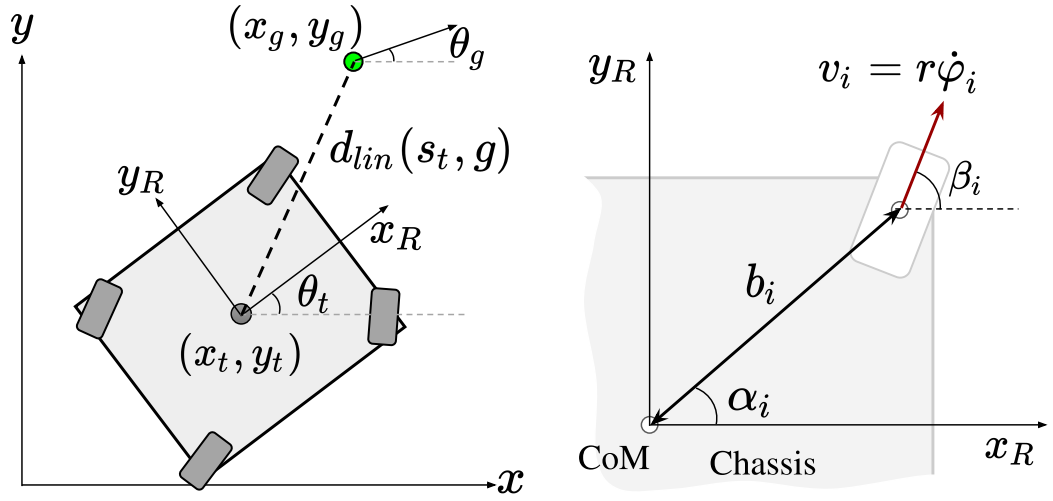
7.3 Problem Statement

The mobile robot

The Wheeled Mobile Robot (WMR) chosen as the subject of this chapter is equipped with four steerable and driving standard wheels, all independent of each other in both direction and speed, also known as *4 Wheel Steering 4 Wheel Driving* (4WS4WD). Such a robot is over-actuated, yet it is subject to non-holonomic constraints due to the use of standard wheels. For this reason, it is also referred to as non-holonomic omnidirectional [139] or pseudo-omnidirectional [134] robot.

Figure 7.3.1(a) shows the coordinate system used in the notation, whereas Figure 7.3.1(b) highlights the relevant notation in terms of actuation variables and geometric parameters referred to the i-th wheel ($i = 1, \ldots, 4$). The pose of the robot is identified by the triple (x, y, θ), i.e. the coordinates of the Center of Mass (CoM) of the robot and its orientation with respect to the world reference frame. The dynamics of the robot are influenced by the following parameters: (i) M, the mass of the robot; (ii) r, the radius of the wheels; (iii) b_i, the length of the segment connecting the CoM and the i-th wheel's steering axis; (iv) α_i, the angle between the x_r-axis and the segment connecting the CoM and the i-th wheel's steering axis; (v) β_i, the wheel's steering angle (a control input); (vi) $\dot{\varphi}_i$, the wheel's angular velocity (a control input).

The high number of controllable wheels and their independence from each other make this robot tailored for a wide variety of tasks, starting from autonomous driving up to fault tolerant control. In fact, the particular locomotion system of this robot allows to obtain the same motion vector through a wide range of actuation solutions, which makes the task of controlling this robot non-trivial.



(a) Notation (b) Chassis

Figure 7.3.1: The considered robot

The goal of this chapter is to design a robust Reinforcement Learning (RL) agent able to regulate the pose of the robot in a plane free of obstacles. Hence, the agent must learn how to drive the 8 motors that govern the motion of the robot to move it towards the desired pose.

Definition of goal g

For each episode a new goal $g = (x_g, y_g, \theta_g) \in \mathcal{G} \subset \mathbb{R}^2 \times SO(2)$ is generated, where x_g and y_g are the Cartesian coordinates with respect to the global reference frame, and θ_g is the desired heading angle (see Figure 7.3.1(a)).

Following the notation introduced in [140], a reward function $r(s_t, g)$ parametrized on g is used. Each goal g is a state belonging to $\mathcal{S}_g \subset \mathcal{S}$, where $\mathcal{S}_g$ is defined as the set of states satisfying the goal g. Equipped with this notation, the agent is said to reach the goal whenever the current state s_t gets in $\mathcal{S}_g$. In the following, $\mathcal{S}_g$ is formally defined.

Let $d_{lin}(s_t, g) = \sqrt{(x_t - x_g)^2 + (y_t - y_g)^2}$ be the linear Euclidean distance between the current state s_t and the goal g; similarly, $d_{ang}(s_t, g) = |\theta_t - \theta_g|$ is the absolute value of the heading error. A graphical representation of the introduced notation is provided in Figure 7.3.1(a). Let $\mathcal{S}_g$ be the set of states satisfying both conditions: (i) $d_{lin}(s_t, g) < \epsilon_{lin}$, (ii) $d_{ang}(s_t, g) < \epsilon_{ang}$, with ϵ_{lin} and ϵ_{ang} being respectively the linear and angular thresholds accepted (tolerances).

To have a compact definition, $\mathcal{S}_g$ is redefined as $\mathcal{S}_g = \{s_t \in \mathcal{S} : \|f(d(s_t, g))\|_\infty < \epsilon\}$ with $d(\cdot)$ being a function returning a vector of two components collecting the linear and angular distances, $f(\cdot)$ being a function that normalizes the linear and angular distances with respect to their tolerances

and ϵ being a threshold on the distance defined over the triple (x_t, y_t, θ_t) with respect to g. Under this notation, in simple terms, the agent reaches the goal g if the state s_t enters the set $\mathcal{S}_g$, i.e. the robot tracks the desired pose with a given tolerance on the Euclidean and angular errors.

Observation and action spaces

The observation space has been designed to be composed of: the linear (e_l) and angular (e_a) errors with respect to the goal coordinates, normalized to their respective initial errors; the ideal velocities in x (V_x^{CoM}) and y (V_y^{CoM}) axes of the CoM of the robot, coherent with the maximal velocity each wheel can impose; each wheel's velocity error with respect to V_x^{CoM} and V_y^{CoM}, defined as $e_{V_x^{CoM}}$ and $e_{V_y^{CoM}}$ respectively; the ideal angular velocity ($\dot{\omega}_t^i$) and the actual velocity ($\dot{\omega}_t$). Thus, the dimension of the observation space is 14.

The action space is composed of the linear velocities and the steering angles of each wheel. The steering angle is represented through its sine and cosine components, following [141]: the representation of a connected set of rotations in a circumference through its angles in the range $[-\pi, \pi)$ introduces a discontinuity. In fact, [141] argues that such discontinuities are typically harder to learn for a neural network. Testing both configurations revealed that this issue still holds. Indeed, representing the rotations directly in radians resulted in an agent incapable of controlling the robot. Also, a representation of an angle through its cosine and sine components allows one to formulate an error on the said angle in the form of a norm.

Reward function design

An extensive phase of *reward shaping* has been conducted and converged to define the reward r_t as the sum of the following components: (i) the linear distance from the goal coordinates $d_{lin}(s_t, g)$; (ii) the angular distance from the goal coordinate $d_{ang}(s_t, g)$; (iii) the sum of the quadratic errors of the wheels' velocities in the x and y directions $e_t^{V_x y}$; (iv) the error of the angular velocity applied with respect to the ideal one $e_t^{\dot{\omega}^i}$. Hence, the reward function r_t becomes parametric with respect to the current state s_t, the epoch's goal g and the action a_t:

$$r_t(s_t, g, a_t) = -d_{lin}(s_t, g) - d_{ang}(s_t, g) - e_t^{V_x y} - e_t^{\dot{\omega}^i} \tag{7.2}$$

The configuration of a unicycle robot is regarded as *ideal*. Defining this guidance for movement enhances exploration efficiency, particularly during the initial stages.

7.4 Methodology

This chapter proposes two tools that aim at improving the exploration, sample efficiency, robustness, and the overall performance of the RL agent. The *Difficulty Manager (DM)* is introduced as a tool to feed the agent with goals always proportioned to its current performance, and the *Episodic Noise*, as a tool to increase the efficiency of the exploration of the action space.

7.4.1 *Difficulty Manager (DM)*

The key idea for the *DM* (Figure 7.4.1) is to create a progression in the difficulty of the goals presented to the agent. The goals g are sampled from a distribution with an average that is increased accordingly to the skills of the current agent. The tail of said distribution will also include all the *simpler* goals, with the intent of preventing the agent from over-fitting on the current distribution.

Hence, in the early phases of training, when the agent is hardly able of moving at all, goals will be generated near the initial pose and, as the agent learns to move efficiently, goals placed further away will be presented to allow the agent improve its performance.

The proposed system is based on three main components as shown in Figure 7.4.1.

Each training episode i produces a couple goal-terminal state, i.e. $(g_i, s^i_{\mathcal{T}})$. The *Episode Classifier* reads this couple and produces a label y_i associated with the current episode to classify its performance. The label y_i is sent to the *Evaluator*, which temporarily stores the label in a queue to observe the evolution of training, and potentially triggers an *evaluation* phase to quantify, by considering the success rate over a set of uniformly sampled goals, the robustness and reliability of the agent; if the model satisfies a certain condition (see below), the current level l is increased and sent to the *Goal Generator*. This component holds the goal distribution $\mathscr{D}_l$ related to the current *difficulty level* $l \in \{1, \dots, L\}$, with L being the maximum difficulty level; for each new training episode it samples a new goal g_i which is sent in input to the new episode.

Episode Classifier. This block maps each episode i to a label y_i that classifies the agent's performance. In particular, the episode can be classified as a *success* ($y_i = \mathrm{S}$, the agent has reduced the distances under the threshold), a *failure* ($y_i = \mathrm{F}$, distances have been reduced but not under the threshold), or a *serious failure* ($y_i = \mathrm{SF}$, the final distances are greater than the initial ones). The classification is performed according to the following conditions on the terminal state $s^i_{\mathcal{T}}$ associated with the i-th goal g_i:

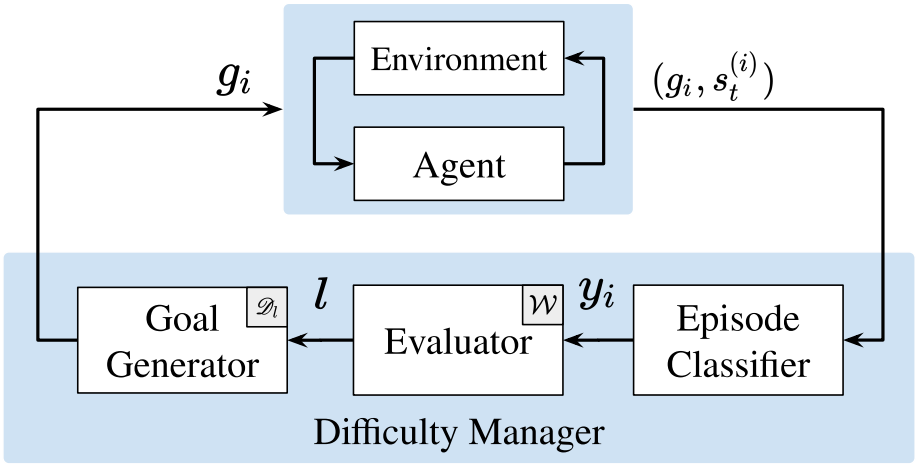


Figure 7.4.1: Difficulty Manager (DM)

$$y_i = \begin{cases} \text{S} & \|f(d(s_t^i, g_i))\|_\infty \leq \epsilon \\ \text{F} & \epsilon < \|f(d(s_t^i, g_i))\|_\infty < (1-\epsilon) \\ \text{SF} & \|f(d(s_t^i, g_i))\|_\infty \geq (1-\epsilon) \end{cases} \tag{7.3}$$

Goal Generator. This component implements a negatively skewed distribution $\mathscr{D}_l$, with an average D_l, associated with the current level l. For each episode, it samples a new goal from the current distribution and sends it in input to the new episode.

Evaluator. This block holds a window of W episodes' labels that allows to observe the leaning evolution. To the purpose, the labels of the latest epochs are stored in a FIFO queue $\mathcal{W}$. The set containing only the labels of episodes in $\mathcal{W}$ classified as *serious failures* or *failures* are respectively defined as $\mathcal{W}^{\text{SF}} = \{y \in \mathcal{W} : y = \text{SF}\}$ and $\mathcal{W}^{\text{F}} = \{y \in \mathcal{W} : y = \text{F}\}$.

Then, the agent accesses an evaluation phase if and only if the following *evaluation criteria* are met: (i) no serious failures have been observed in the current window (i.e., $|\mathcal{W}^{\text{SF}}| = 0$) and (ii) the ratio of episodes rated as *failed* ($y_i = \text{F}$) is lower than a given threshold η: $|\mathcal{W}^{\text{F}}|/|\mathcal{W}| < \eta$.

During the evaluation, the model is tested on a set of different goals generated uniformly in the range of distances reached so far. The objective is to save a robust model (π_θ^l, π_Q^l), associated with the current level l, that will be carefully evaluated later. The robustness is evaluated considering the following conditions: (i) the successful tests must exceed a given threshold T (i.e., the number of failures must be below the complementary threshold $1 - \text{T}$); (ii) no serious failures must be registered. When an evaluation phase concludes positively, the model is saved, the level is increased by one ($l \leftarrow l + 1$) and the window is emptied.

Clearly, the first models to be saved only saw the initial distributions and, as a consequence,

will hardly be capable of generalising to goals placed at higher distances, however, it is interesting to analyze the different behaviors that each saved model exhibits. More insights are presented in Section 7.5.

7.4.2 *Episodic Noise*

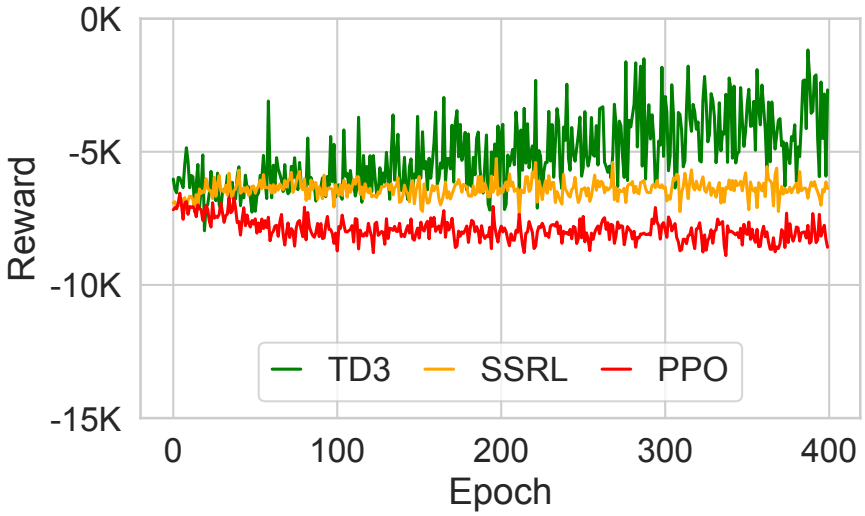
The second proposed tool concerns the noise applied over the actions. In this chapter, an "*episodic noise*" has been used to improve the exploration of the action space, in contrast to the typical noise identified by a zero mean Gaussian (or uniform) distribution. One of the main reasons behind this choice is that completely random actions (especially in the initial phases) may cause the agent to barely move from the initial position, due to the peculiar dynamics of the robot. Instead, for each episode, a value ρ is sampled from a Gaussian distribution, clipped in the range $(-1, 1)$. Such value is then used at each step of the current episode as the mean of a new distribution with standard deviation σ (a hyperparameter), from which the actual noise is sampled. This generates a noise that, inside the episode, keeps around the set average (most likely different from 0), allowing the appearance of more useful and continuous behaviors than the ones obtained from noise with zero mean. As the agent learns, this signal is reduced in order to make more space for exploitation.

7.5 Results

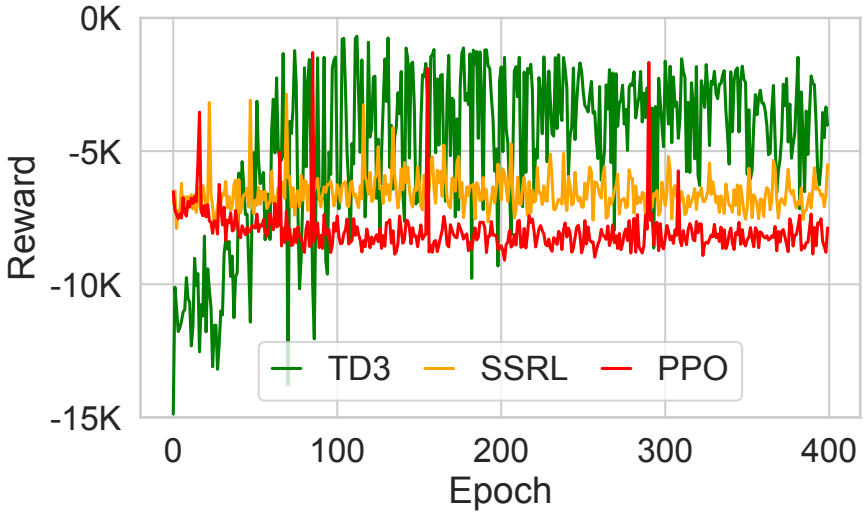
This section compares three state-of-the-art RL algorithms (PPO, SSRL, and TD3), and then focuses on an ablation study of the proposed tools on the best-performing algorithm. The results shown in the following have been obtained by implementing the robot and environment described in Section 7.3 in "*Gym Ignition*" [142]. Such a simulation environment has a light physics engine and benefits from the interfaces of "*OpenAI Gym*" [143]. This simplifies the process of creating new environments and tasks.

Figure 7.5.1 shows the results of the three agents with (Figure 7.5.1(b)) and without (Figure 7.5.1(a)) our proposed tools.

Even though the proposed tools allow all the considered algorithms to improve performance in the first stages, neither PPO nor SSRL manage to solve the task over 600k interaction timesteps. Two key factors are identified in relation to these negative results. Being PPO an on-policy RL algorithm, during training it cannot make use of meaningful episodes (the ones during which the robot moves properly, which are rarer when the agent is still learning) as much as off-policy algorithms do, since the latter are equipped with a replay buffer. Regarding SSRL, the main limitations might be ascribed to the update technique and the buffer employed: in the initial phases the agent will most likely



(a) Baseline Agents



(b) Agents with Ep.N and DM

Figure 7.5.1: RL Algorithms Comparison

experience trajectories associated with poor performance and on this experience it employs imitation learning that leads toward suboptimal updates, falling in a vicious cycle that impairs significant improvements. Note that, in Figure 7.5.1 both PPO and SSRL equipped with the proposed tools show some improvements (several peaks can be observed) but they fall on the same pattern soon after epoch ~ 100. Even if our tools slightly improve (at least in the beginning) the performance of all the considered agents, they are not meant to solve issues related to the algorithms themselves, but rather to improve their sample efficiency.

Figure 7.5.1(a) shows that TD3 baseline is able to improve performance but its performance is not robust. However, by leveraging the proposed tools, TD3 reaches good performance in less than 100 episodes (less than 150k interaction timesteps) and stabilises over time, even on more complex goals (Figure 7.5.1(b)).

For this reason, in the remaining part of this section, the study is narrowed down to TD3. In particular, four agents trained with the TD3 algorithm are inspected: (i) *Baseline* (BL), which is the vanilla TD3; (ii) *Baseline with DM* (BL+DM): which is the vanilla version equipped only with the DM tool; (iii) *Baseline with Episodic Noise* (BL+Ep.N), which is the vanilla version equipped only with the Episodic Noise tool; (iv) *Baseline with DM and Episodic Noise* (BL+DM+Ep.N, also referred to as "Ours" for brevity purposes), which is the vanilla version equipped with both the tools[1].

Table 7.1 reports the key hyperparameters of the algorithm and the proposed tools. It is worth mentioning that both the actor and critic approximators are composed of two hidden layers, 512 and 256 respectively. All tests consist of reaching the same 200 goal coordinates for each agent, generated uniformly in a range of 6 meters for the Cartesian coordinates and in the whole range of $[-\pi, \pi)$ for the angular goal. Note that, during training, which lasts at most 2000 epochs, all agents will experience goals distant at most 4 meters away. Hence, the performance of the model will also be evaluated on greater distances than the ones experienced during training, in order to assess the generalization capabilities of the considered model.

Note that, in all tests conducted, the BL agent has never met our *evaluation criteria*. However, for a comprehensive comparison of all the agents, the BL agent's models presented have been saved every 500 epochs.

[1] https://c3lab.poliba.it/SERL provides videos showing the behaviour of the trained agents as they progress through their training phases

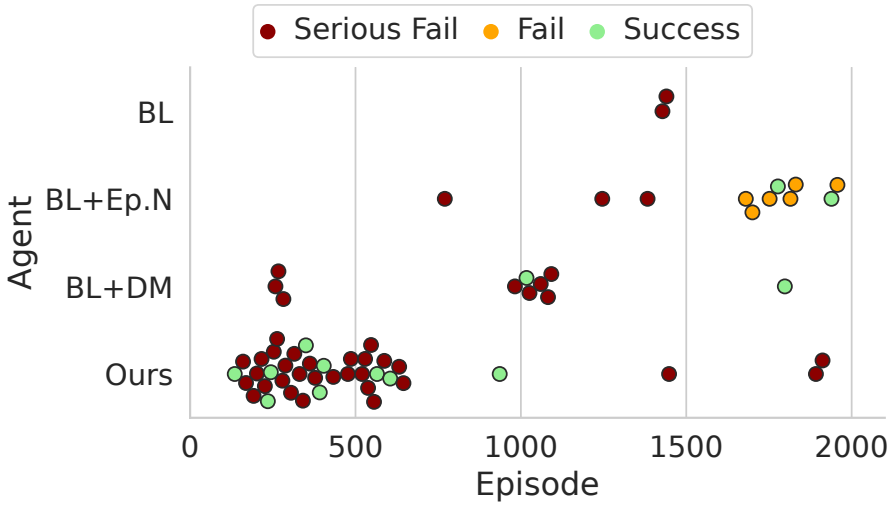
Table 7.1: Hyperparameters

	Hyperparameters	Value
Agent	LR Actor	0.0001
	LR Critic	0.0001
	γ	0.95
	τ	0.1
	σ	0.3
	Noise Decay	0.99
	Actor Update Rate	125
	Batch Size	32
Memory	Type	PER
	Mem. Capacity	50000
	α	0.5
	β	0.4
DM	W	10
	η	0.1
	Init. Dist. (m)	0.6
	Step Dist. (m)	0.4
	Init. Ang. Dist. (rad)	0.75
	Step Ang. Dist. (rad)	0.6
	Max Goal Dist. (m)	4.0
	T	0.85

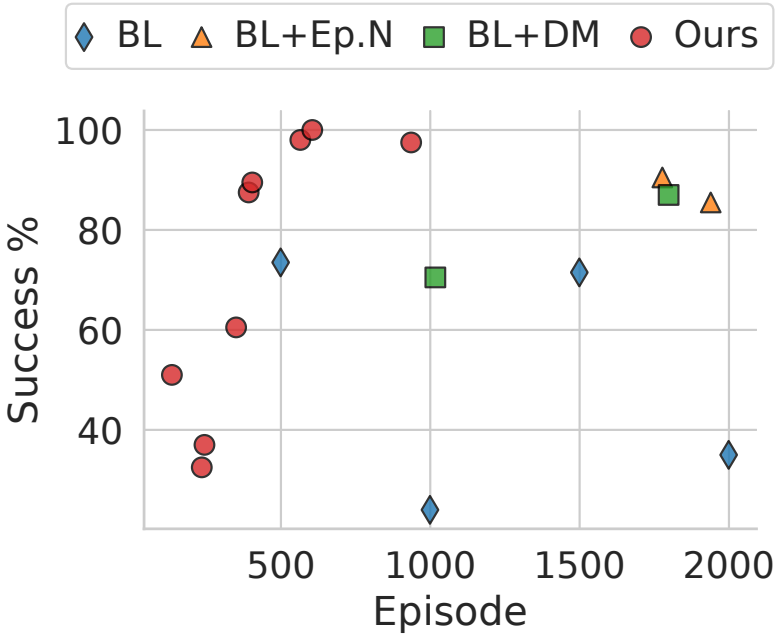
7.5.1 Overall Performance

In order to assess the impact of each component, an evaluation of the performance of each agent during training is carried out. Figure 7.5.2(a) reports every access to the evaluation phase and the corresponding episode at which such evaluation occurred. Each evaluation's result is reported and represented by a $\circ$, differentiating between successful evaluations (green $\circ$), failing evaluations (orange $\circ$) and evaluations interrupted by a serious fail (red $\circ$). Notice that evaluations are grouped by agent (the y-axis of Figure 7.5.2(a)). As an overview of the performance of each considered agent, Figure 7.5.2(b) shows the success rates obtained during testing (expressed in percentage) for each model that succeeded an evaluation in the sense defined in Section 7.4.1.

As one can observe from both Figure 7.5.2(a) and Figure 7.5.2(b), the baseline (BL) does not really show a progression: the success rates of the sampled models are modest and quite variable, indicating poor robustness of the trained policy. Moreover, BL shows no progression as training is carried out. The BL+Ep.N obtains good performance (85-90% success rate) only after $\sim$1750 episodes of training, producing two "*good models*", i.e. models that pass the evaluation phase. The BL+DM passes only two "levels", experiencing goals at a maximum average of $\sim$1.5 meters. The first model (obtained quicker than BL and BL+Ep.N) shows good generalisation capabilities,



(a) Evaluations per Agent



(b) Tests' Success Rate

Figure 7.5.2: Overall Performance

especially considering the (simpler) goals experienced. The second model shows slightly increased performance. Finally, our agent ("Ours") shows a particularly interesting trend: the first good model to pass the evaluation is obtained very quickly (epoch 135) and is followed by a condensed sequence of evaluations (successful in many cases), highlighting how the tools help the algorithm steadily improve performance towards convergence. The peak in performance is registered at epoch 605 with a success rate of 100%. In summary, the proposed agent ("Ours") improves the sample efficiency compared to the other agents that lack the two tools.

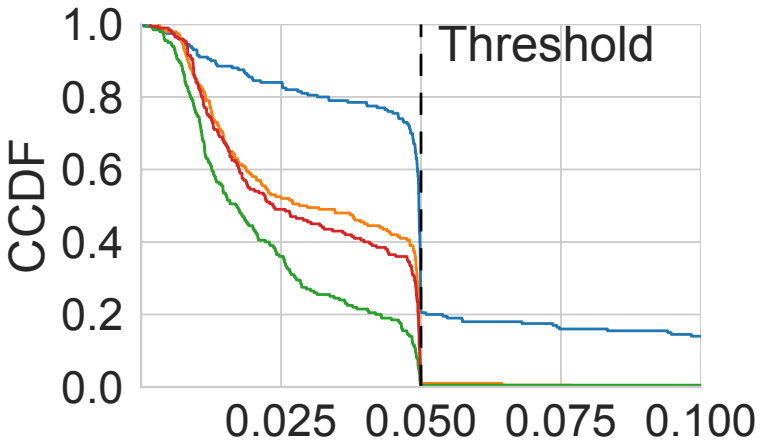
A first intuition is related to the goal distance: providing goals far from the origin during the early training stages impairs the efficiency of the training process. This is due to a vicious circle triggered by three main factors: (i) in the early stages the agent cannot produce significant movements; (ii) hence, the observations in the replay buffer will not be characterized by a wide diversity (since the agent keeps close to the origin), thus possibly producing updates that lead to local sub-optimal policies and (iii) as a further consequence, the rewards associated to these state-action pairs will be quite similar to each other, creating a valley in the cost function. In order to escape this local minimum, the agent should pseudo-randomly produce at least a barely good trajectory (a situation that is more likely to occur when the sampled goal is close enough to the origin). Providing closer goals increases diversity in the buffer, due to a reduced magnitude of the errors, allowing the agent to repel ineffective actions and select much more frequently those actions associated with higher rewards.

7.5.2 Linear and Angular Distances Coverage

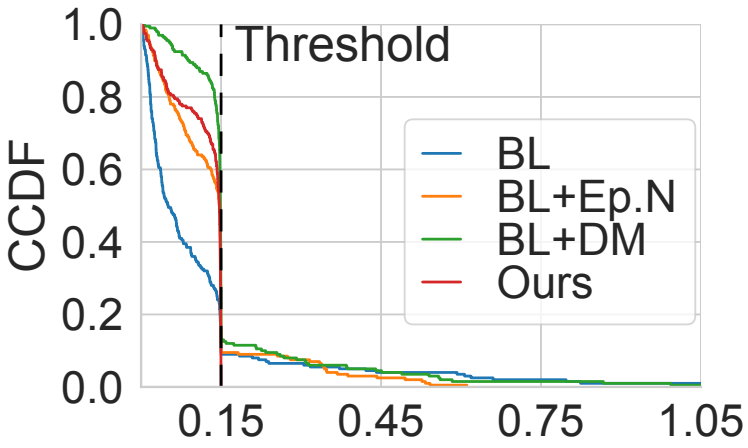
Next, the robustness of the best models obtained for each agent is investigated. Figure 7.5.3(a) and Figure 7.5.3(b) represent the complementary cumulative distribution functions (CCDFs) of, respectively, normalized linear distance and normalized angular distance. Both figures show a dashed vertical line which represents the thresholds for linear and angular distances. Only those models whose CCDFs are below both the thresholds are considered to have reached good performance, see Section 7.3.

Observing Figure 7.5.3(a), it may seem that the BL+DM agent (in green) achieves the best performance; however, a closer look reveals that it does not grant the satisfaction of the success condition, since the curve slips beyond the threshold. On the contrary, the agent equipped with both the proposed tools (in red) grants complete coverage, never trespassing the threshold.

Figure 7.5.3(b) clearly shows that only the agent equipped with both the tools (in red) is capable of respecting the thresholds in all the cases. The second best performing agent is the baseline equipped with the episodic noise, which grants on $\sim$90% the complete coverage of the angular



(a) CCDF by Normalized Distance



(b) CCDF by Heading Error [rad]

Figure 7.5.3: CCDFs on linear and angular distance

distance.

7.5.3 Maneuverability Exploitation

Another interesting result regards the ability of the agent of fully exploiting the robot's maneuverability: agents equipped with the Episodic Noise learn to balance the minimization of linear and angular distances throughout the epoch, while agents without such tool learn to minimize either linear or angular distance first and only later they minimize (or try to) the other metric.

7.5.4 Behaviour's Analysis

A crucial aspect in controlling the chosen mobile platform is represented by the ability of the controller of fully exploiting the motion capabilities of the robot. For this reason, it is interesting to study how each agent learns to control the robot[2].

In Figure 7.5.4 some of the most representative trajectories for each agent, during tests, are shown. Due to space constraints, only the successful trajectories of the best models for each agent are considered. The trajectories are divided into two categories: the ones in which the condition over the linear distance is satisfied first (represented in blue), and the trajectories in which the agent prioritizes the angular distance (displayed in orange). This categories represent the behaviour the agent leverages to reach the goal.

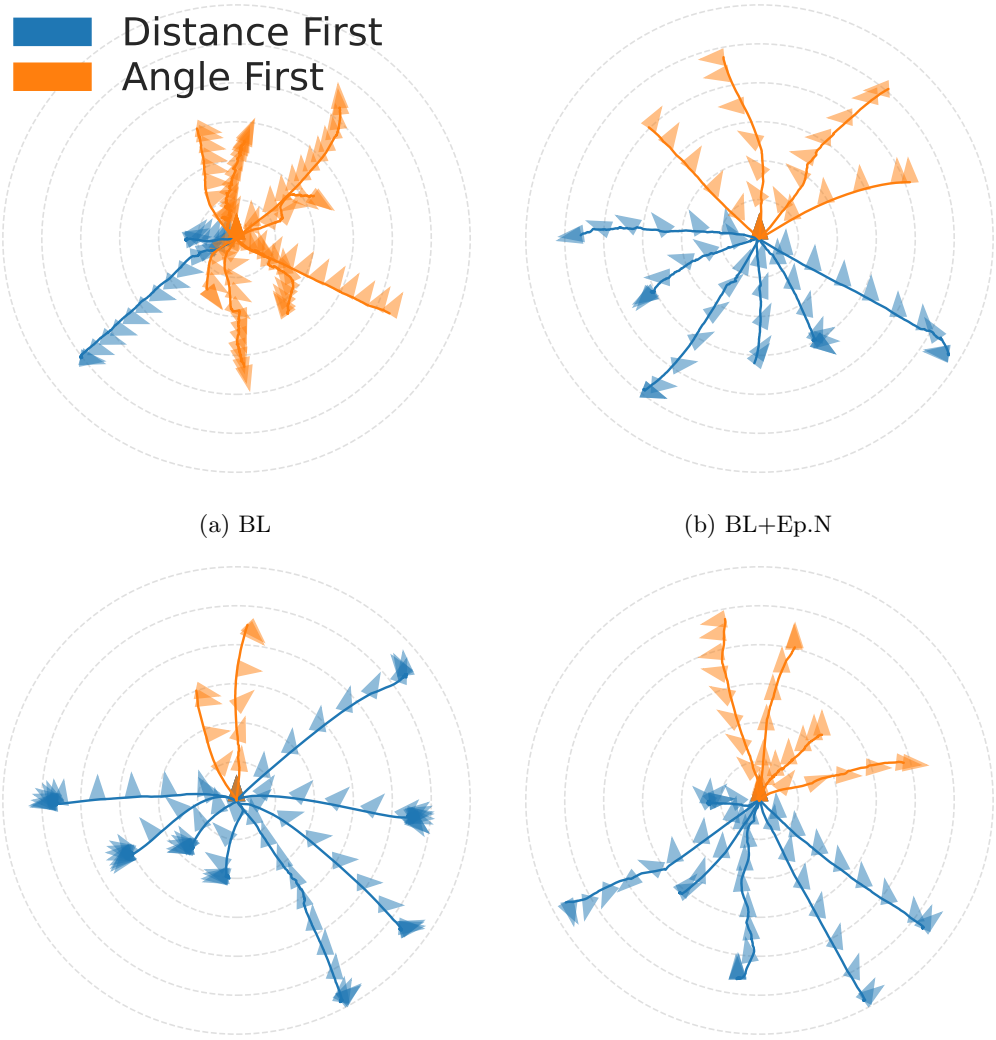
7.5.5 Discussions

This section provides a discussion of the key results obtained.

Regarding the tools provided, the Episodic Noise improves the exploration phase, allowing complex behaviours to emerge already in the first (and crucial) episodes; the DM provides, during training, goals appropriated to the current abilities of the agent, letting the agent see only the episodes which are truly relevant to the training.

The impact of these tools is quite noticeable: the agent shows a steady improvement of the performance already in the first few hundreds epochs, entering the evaluation phase many times and peaking in performance near epoch 600, while the baseline by itself does not meet our *evaluation criteria* in the 2000 episodes of training. This highlights how the tools provided allow to greatly improve the sample efficiency and the exploration of the baseline algorithm. Furthermore, the behaviours learnt are analysed with the complementary cdfs (Figure 7.5.3) of the linear and angular distance at the last timestep of each test and the actual paths followed by the agent (Figure 7.5.4).

[2]https://c3lab.poliba.it/SERL provides video showing the behaviour of the trained agents as they progress through their training phases



(a) BL

(b) BL+Ep.N

(c) BL+DM

(d) Ours

Figure 7.5.4: Representative successful trajectories

The agent shows a complete coverage of both linear and angular distances, but also a smoother trajectory. This last detail is important in view of applying such controllers in real scenarios, where one should also consider the trajectories and their smoothness. Also, one could also consider evaluating the non-holonomic constraints of the robot (no sliding and no slipping conditions on the wheels). This is particularly interesting and can be further analysed in future works.

An interesting work could see this agent employed in a hierarchical architecture in which the agents in a higher level are removed from the tedious and complex task of directly controlling the robot itself and can focus on more complex tasks such as image recognition and obstacle avoidance. The higher level policies will be in charge of producing the next goal position or the set of linear and angular velocities that the lower agent should follow, which are significantly simpler to learn compared to the inverse kinematic model of the robot.

A final thought on an interesting application is presented. Thanks to the high number of actuators, an interesting application would be the *fault tolerant control*: such a number of actuators could manage, with an appropriate controller, to complete a desired task (f.i. pose regulation) including scenarios in which one or more actuators undergo a fault.

7.6 Concluding Remarks

This chapter studied the control of an over-actuated mobile platform for a pose regulation task. Unlike the classical approach, which usually limits the maneuverability of the mobile platform, a learning-based, model-free approach is used. The performance of some of the most popular RL algorithms are studied. An in-depth analysis on the TD3 showed promising results, despite the base algorithm never meeting the set requirements. Hence, two new tools are introduced: the Episodic Noise and the Difficulty Manager (DM). The combination of these two tools improves the performance of the base algorithm, outperforming it and reaching 100% success rate during tests, including goals never experienced.

Chapter 8

Safe Reinforcement Learning for Autonomous Navigation of a Driveable Vertical Mast Lift

8.1 Introduction

The fourth industrial revolution (Industry 4.0) is changing the way products are manufactured, logistic is executed, and services are delivered to customers [144]. The deep integration of Cyber-Physical Systems (CPS) in business operations is boosting innovation in this area, leading to significant improvements in efficiency and productivity [145]. Autonomous mobile robots represent a key asset for the implementation of the Industry 4.0 paradigm. In fact, mobile robots are being increasingly adopted in a wide range of applications ranging from more classical ones, i.e., logistics, to more complex operations requiring robots to cooperate with both other robots and humans, typically in unstructured environments [146]. In such domains, safety becomes a relevant issue to address: damage to the equipment may result in loss of profit, while damage to humans may cost lives. Such domains, also known as *Safety Critical* CPS, are associated with a number of *safety standards* that aim at preventing the occurrence of those accidents [147].

In order to make mobile robots autonomous, tools should be designed to allow perception of the environment with sufficient details to feed navigation control algorithms. To this purpose, robotic systems can benefit from Artificial Intelligence (AI) techniques that are showing very promising results in diverse engineering fields [148]. In particular, Reinforcement Learning (RL) is an AI data-driven approach which is being increasingly adopted in industrial applications to address complex control tasks [149]. The behavior of a RL control policy, i.e. how the agent acts based

on the observations of the environment E in discrete time steps t, is similar to the operation of a controller in a classical feedback control system [150]. Classical RL approaches typically assume that the task can be modelled as a Markov Decision Process (MDP), consisting of the set of states, the set of actions, the rewards and the transition probabilities that capture the dynamics of a system. In such a configuration, the policy must learn the actions that optimize the reward directly from its experience, which is collected through a trial and error process, namely *training* [151]. While this approach often works satisfactorily well in proof-of-concept environments, constraint violations could not be tolerated in scenarios regulated by safety standards and laws. It is, then, important to ensure that the developed control systems are safe throughout the entirety of its employment, hence during both training and deployment [152, 153].

In order to ensure safety constraints, especially in scenarios regulated by safety standards and laws, a recent RL approach is proposed, namely Safe Reinforcement Learning (Safe RL). Safe RL embeds safety constraints into the RL problem formulation to ensure safety during both the learning phase and the deployment (see [154]). Such techniques adopt several strategies to embed safety into the whole process. A popular approach extends the reward function with signals related to constraint violations, but cannot grant strict compliance of the policy to the constraints, especially during the learning phase [155]. Another popular approach leverages an extended version of the MDP framework that includes constraints in the formulation, namely Constrained Markov Decision Process (CMDP). This framework ideally restricts optimization to the *safe space* that should respect the constraints. In practice, however, such a formulation might produce poor policy performance or constraint violations [156, 157].

An investigation on the advancements in the field of Safe RL is carried out from a practical perspective. Several state-of-the-art algorithms are employed to equip an Aerial Mobile Lifting Working Platform (AMLWP) with autonomous driving capabilities, while safeguarding constraints. In particular, these algorithms are compared on the task of controlling a Driveable Vertical Mast Lift (DVML), a particular configuration of AMLWP [158]. Such machines are particularly heavy, with weights in the range of several hundreds of kilograms. The DVML has been carefully modelled to faithfully represent an actually employed robot. This application allows to highlight critical aspects regarding the gap between the design of such algorithms and their deployment in real use case scenarios, where safety standards are involved. In fact, these algorithms are often tested in simpler environments that work as proof-of-concept and tend to neglect important aspects of real machines and scenarios. A DVML has been selected since it is suitable to represent a safety critical scenario. This chapter does not limit the evaluation of the performance of the policy in reaching a desired goal position while avoiding collisions, but it also considers the intensity of vibrations during

the operations (and possible tip-overs).

The remainder of this section is organized as follows. Section 8.2 presents a review of the related work. Section 8.3 analyzes the main aspects of the algorithms used. Section 8.4 describes the problem statement and the resulting design choices to accurately model the task. The experimental results and their critical evaluations are, then, provided in Section 8.6. The chapter ends with some discussions related to the main insights and findings in Section 8.7.

8.2 Related Work

All RL algorithms considered in this chapter can be seen as variations of three popular Deep Reinforcement Learning (DRL) algorithms: Deep Deterministic Policy Gradient (DDPG) [38], Proximal Policy Optimization (PPO) [25], which can be seen as a natural extension of Trust Region Policy Optimization (TRPO) [36] methods, and Soft Actor Critic (SAC) [37]. Such algorithms share an Actor-Critic agent's structure, in which the critic's role is to approximate the loss function, while the actor interacts with the environment while optimizing the performance with respect to the task. The main differences are briefly summarized below.

Off-Policy vs. On-Policy: while PPO, being an On-Policy algorithm, needs to apply updates based on the current state of the policy, therefore not being able to reuse experience from previous episodes, SAC and DDPG can make use of a Replay Buffer to leverage experience collected throughout the training.

Policy Mapping: both PPO and SAC share a probabilistic actor which maps states to a distribution probability over actions ($\pi : \mathcal{S} \rightarrow P(\mathcal{A})$), while DDPG employs a deterministic policy which directly maps states to actions ($\pi : \mathcal{S} \rightarrow A \in \mathbb{R}$).

Value function approximation: all three algorithms adopt a distinct approach. PPO approximates the State-Value function ($\mathcal{V}(s_t)$), DDPG approximates the Action-Value function ($\mathcal{Q}(s, a)$) while SAC approximates an Action-Value function and a Soft State Value function $\mathcal{V}(s_t) = \mathbb{E}[\mathcal{Q}(s_t, a_t) - \log \pi(a_t|s_t)]$.

Task Objective: while PPO and DDPG try to maximize the expected sum of rewards, SAC augments the objective with an entropy term to improve exploration:

$$\sum_t \mathbb{E}[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))] \tag{8.1}$$

Note that while the expected sum of rewards is the same for PPO and DDPG, the first adopts a very different loss function, which has a clipped form and can adopt a Kullback-Leibler divergence term, depending on the implementations.

In the following is presented the related literature that focuses on two main aspects: 1) the RL methodologies that have been proposed for safety critical applications; 2) the application of RL to industrial scenarios.

8.2.1 RL Methodologies for safety critical applications

[26] adopt a Safety Layer in order to compute an action correction in a closed loop form. However, it is not enough to guarantee constraint satisfaction at training. [159] and [160] solve the CMDP optimization problem through Lyapunov optimization. This approach is able to grant near-constraint satisfaction, by mapping actions (or policy parameters) into a safe set. [161] compute a Forward Reachable Set by leveraging an uncertain dynamical model of the robot. If the action is considered to be unsafe, it is discarded in favour of a previously computed safe action. [162] make use of Imitation Learning from an expert. The problem solved in such work has a discrete action set. Note that, in general it is not possible to leverage an expert and, especially with complex dynamics, a discrete action set may not be sufficiently expressive. [163] combine Dynamic Window Approach with DRL to improve navigation performance but do not guarantee constraints satisfaction during training. In Reinforcement Learning for Formation Control of Mobile Robots, [164] employ RL based Model Predictive Control (MPC) in a distributed fashion to coordinate a fleet of robots. While they simplify the optimization phase thanks to the RL approach, they still need a model of the environment. [165] employ an RL agent at a lower sampling rate in order to obtain intermediate goals. Safety is tackled through a *safety shield* which evaluates the trajectory and produces corrections to grant constraint satisfaction. The approach is proved on a manipulator application. [166] propose a new RL algorithm based on *Trust Region methods* which approximates the constraints satisfaction (or violation) variation caused by each update, in order to guarantee a constraint satisfaction within a certain threshold, which is an hyperparameter, namely Constrained Policy Optimization (CPO). [167] tackle the problem of Safe RL by estimating a *safety certificate* of the current state. A safety certificate is a measure of how safe the current state is: higher values are associated with states closer to a constraint violation. A safe control policy is trained to learn to keep the safety certificate to a low value, i.e. granting the constraint satisfaction. [168] propose an Implicit Safety Set Algorithm (ISSA) which employs an optimization algorithm, namely Adaptive Momentum Boundary Approximation Algorithm (AdamBA), in order to guarantee that each action satisfies the constraints. [169] introduce a Constrained Variational Policy Optimization (CVPO) algorithm, which leverages a distribution of safe trajectories (approximated while learning) and train the agent in an Expectation-Maximization fashion to produce trajectories belonging to said distribution. [170] propose a two-stage algorithm called First Order Constrained Optimization in Policy Space (FOCOPS) that initially searches

for the optimal policy update by solving the CPO policy update optimization problem into the nonparametrized policy space. Then, first-order methods are employed to minimize a loss function formulated to project the first-stage policy update back into the parametrized policy space. [171] present a primal approach, namely Constrained Rectified Policy Optimization (CRPO), which initially updates the policy without regard to the constraints, but if such an update results in a constraint violation, the policy is rectified to minimize a function of the constraints descending toward the direction in which it has been violated. [172] extend PPO into the so-called Penalized Proximal Policy Optimization (P3O) that introduces a penalization term into the objective function via Rectified Linear Unit (ReLU) operators, leading to an unconstrained problem. In this way, if the agent violates a constraint, the penalized term dominates; if not, the objective consists in the standard policy optimization. [173] propose a two-step approach called Constrained Update Projection (CUP). First, an optimization algorithm improves the expected cumulative reward, while minimizing the distance between old and current policy. Second, the policy obtained during the first step is projected onto the safe constraint set to counter any constraint violation. [174] propose the Safety AUgmenTEd (SAUTÉ) MDP formulation so that safety is taken into account by augmenting the state space using the residual safety budget introduced in [175] (in the so-called Safe policy IMproveMEnt for RL (SIMMER) approach) to determine the constraint satisfaction for each training step. Then, such a function is used to reshape the objective, which consists of an unconstrained minimization problem. [176] address constraint satisfaction by learning a neural barrier certificate function. Specifically, the loss function associated with the barrier certificate consists of three terms, and that representing the invariant property of the barrier is included as a constraint in the policy optimization problem formulated using the Lagrangian dual form. [177] present an augmented Lagrangian-based algorithm, namely Augmented Proximal Policy Optimization (APPO), which leverages a quadratic penalty term to enrich the formulation of the constrained problem using the primal-dual mode. Such a penalty influences the update of Lagrangian multipliers with the aim of reducing constraint violations. Then, the policy update is performed according to the PPO-based clipped technique.

8.2.2 RL for Industrial applications

[178] propose a Q-learning-based approach for cooperative handling operation in industrial multi-robot stations. While the agent takes discrete actions to move from one position to another, safety is granted through a system of priorities between the operating robots: the robot with a currently lower priority is forced to move in a “safe” direction, computed as the shortest path. [179] employ Q-Learning to solve the wire-loop task (a classic game) with a six-degree-of-freedom robot. The

agent obtained shows good generalization capabilities, but safety is not taken into account. [180] evaluate DDPG and PPO algorithms for learning picking operations using a mobile manipulator. The controller must drive the mobile base in a given position to allow the manipulator to reach a second goal. Safety is not taken into account, and the simulation is interrupted if a collision is detected. [181] employ a DDPG agent in order to control an industrial robotic arm which must follow a randomly generated trajectory on a bidimensional surface, simulating an automatic quality inspection. The agent computes actions in the form of position variation. In this case, safety is not considered. [182] synthesize a control policy (trained with TRPO algorithm) for the manipulation of a hydraulic excavator with non-linear dynamics, but neglect the safety aspects. [183] extend the previous work by including the control of the shovel at the end of the arm. They also improve the controlability and usability of the said robot. [184] address the problem of ensuring the safety of a human operator when interacting with an industrial robot by combining a DDPG algorithm with the optimization of an additional intrinsic reward function.

8.3 State of the art

This section highlights the main aspects of the considered Safe RL algorithms. In particular, the focus is set on the following algorithms:

Lagrangian Methods expand the optimization problem, with $f(\theta)$ being the objective, in constrained contexts by adding a penalty coefficient $\lambda g(\theta)$, with λ being the Lagrangian multiplier. The problem to solve becomes a max-min optimization problem: $\max_\theta \min_{\lambda \geq 0} \mathbb{L}(\theta, \lambda) \doteq f(\theta) - \lambda g(\theta)$. [185] implement a Lagrangian version of PPO and TRPO algorithms. Another implementation is proposed by [186]. In particular, they take a control perspective on the Lagrangian methods and leverage the constraint function derivatives to implement a PID controller. Therefore, the Lagrangian multiplier variation can be written as: $\dot{\lambda} = \alpha g(\theta) + \beta \dot{g}(\theta) + \gamma \ddot{g}(\theta)$, with each term being composed by a coefficient and, respectively, the integral, the proportional and the derivative of the constraint function.

Constrained Policy Optimization (CPO, [166]) is a trust region method for Safe RL which approximately enforces the constraints in every policy update. Thanks to this approximation, CPO is able to predict the changing in constraint costs at the end of the specific update and, based on this estimation, the algorithm chooses the update which allows to have constraint costs within a given range, while improving task performance. The authors show that the proposed methodology can train neural network policies with a large number of parameters in computationally complex

constrained control tasks. This is possible by solving the CRL optimization problem:

$$\begin{aligned} & \theta_{k+1} = \arg\max_{\theta} g^T(\theta - \theta_k) \\ \text{s.t.} \quad & c_i + b_i^T(\theta - \theta_k) \leq 0 \quad i = 0, \dots, m \\ & \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \leq \delta \end{aligned}$$

using its dual formulation. Note that g represents the gradient of the objective function, c_i and b_i are respectively the i-th constraint and the gradient of the i-th constraint, H is the Hessian of the KL-divergence $D_{KL}(\pi, \pi_k)$. The authors augment the optimization problem with a term that bounds to a step-size δ the weights updates, in order to limit the error due to linearization around π_k. When the optimization step leads to an unfeasible optimization problem, a backtracking algorithm is employed that updates the policy just to decrease the constraint value.

Implicit Safe Set Algorithm (ISSA, [168]) exploits the Safe Set Algorithms (SSA, [187]) in order to guarantee the safety of the system when a given learning algorithm is employed. The algorithm employs a rule to synthesize proper safety indexes that always grant the existence of a safe control input over the whole set of states. When employing the policy, they use a black box approach to project the nominal action a_t^r on the set of safe control inputs $\mathcal{U}_s(x)$, by solving the following optimization problem:

$$\begin{aligned} & \min_{a_t \in \mathcal{A}} \|a_t - a_t^r\|^2 \\ \text{s.t.} \quad & \phi(f(s_t, a_t)) \leq \max\{\phi(s_t) - \eta, 0\} \end{aligned}$$

where ϕ is the safety index function. In order to solve this problem, authors propose an algorithm called *Adaptive Momentum Boundary Approximation Algorithm* (AdamBA). Starting from the nominal control action, AdamBa performs a linear search to find the boundaries of the safe set. Unit gradient vectors that sample the control input set are exponentially increased until the boundaries are reached. Gradient vectors that do not reach such boundaries are discarded. Then, an exponential decay is performed to find the boundary points of the safe control space. The final output action is chosen with respect to the minimum deviation from the nominal action. Note that authors do not make any assumptions on this method and are, therefore, able to leverage such technique on a wide variety of RL algorithms.

Constrained Variational Policy Optimization (CVPO, [169]) uses an Expectation-Maximization approach to include safety during training. In particular, an optimality variable $\mathcal{O}$ is introduced to represent the event of a trajectory τ maximizing the expected reward. Denoting with $p_\pi(\tau)$ the

probability of following the trajectory τ under the policy π, it is shown that the log-likelihood of optimality under policy π, defined as $\log p_\pi(\mathcal{O}=1)$ is bounded as follows:

$$\log p_\pi(\mathcal{O}=1) \geq$$
$$\mathbb{E}_{\tau\ q}[\sum_{t=0}^{\infty}\gamma^t r_t] - \alpha D_{KL}(q(\tau)||p_\pi(\tau)) = \mathcal{J}(q,\pi)$$

with $\mathbb{E}_{\tau\ q}[\sum_{t=0}^{\infty}\gamma^t r_t]$ being the expected infinite discounted reward, $D_{KL}(q(\tau)||p_\pi(\tau))$ being the Kullback-Leibler distance between an auxiliary trajectory distribution $q(\tau)$ and α being a temperature parameter. To guarantee the satisfaction of the constraints, $q(\tau)$ is chosen from the feasible distribution set:

$$\Pi_{\mathcal{Q}}^{\epsilon_1} := \{q(a \mid s) : \mathbb{E}_{\tau\sim q}[\sum_{t=0}^{\infty}\gamma^t c_t] < \epsilon_1, a \in \mathcal{A}, s \in \mathcal{S}\}$$

with c_t being the cost and ϵ_1 being a threshold. Then, q is updated during training in order to maximize $\mathcal{J}(q,\pi)$ during the expectation step. During the Maximization step, the policy parameters θ are updated with respect to the objective:

$$\bar{\mathcal{J}}(\theta) = \mathbb{E}_{\rho_q}\left[\alpha\mathbb{E}_{q_i^*(\cdot|s)}\left[\log \pi_\theta(a \mid s)\right]\right] + \log p(\theta) \tag{8.2}$$

where ρ_q is the stationary state distribution induced by q. This algorithm can be applied in an off-policy setting by approximating ρ_q through samples collected in the replay buffer. The authors show performance improvements in terms of sample efficiency compared to other state-of-the-art algorithms.

8.4 Problem statement

The inertial properties of the model are tuned to reproduce the actual properties of a real DVML: the basket motion along the vertical axis is achieved with a number of *actuators* on the boom; the boom's vibrations are simulated through an ad-hoc *mass-spring-damper system* setup; the Differential Drive (DD) locomotion mechanism is implemented using two *PID actuators*, one for each driving wheel. The simulation environment used in this chapter is *MuJoCo* [188], an accurate open source framework for physics simulations. In particular: a custom actuator has been implemented to reproduce the PID controllers; the boom has been implemented with three components, the first

Table 8.1: Weight for each DVML item.

Item	**Weight (kg)**
Basket	150
$C_{(1\|2\|3)}$	16
Base	500
Single wheel	10

Table 8.2: Joints configuration for our custom MuJoCo model.

Movement	Joint	Type	Axis	Damping ($N\frac{s}{m}$)	Stiffness ($\frac{N}{m}$)	Springref	Actuator attached
Column vibration	$C_1 - C_2$	slide	0 1 0	730	1560	0	None
	$C_1 - C_2$	slide	1 0 0	730	1560	0	None
	$C_2 - C_3$	slide	0 1 0	665	1350	0	None
	$C_2 - C_3$	slide	1 0 0	665	1350	0	None
Basket vertical elevation	$C_1 - C_2$	slide	0 0 1	None	None	0	Position
	$C_2 - C_3$	slide	0 0 1	None	None	0	Position
Differential drive	Back right wheel	hinge	0 1 0	None	None	0	PID
	Back left wheel	hinge	0 1 0	None	None	0	PID

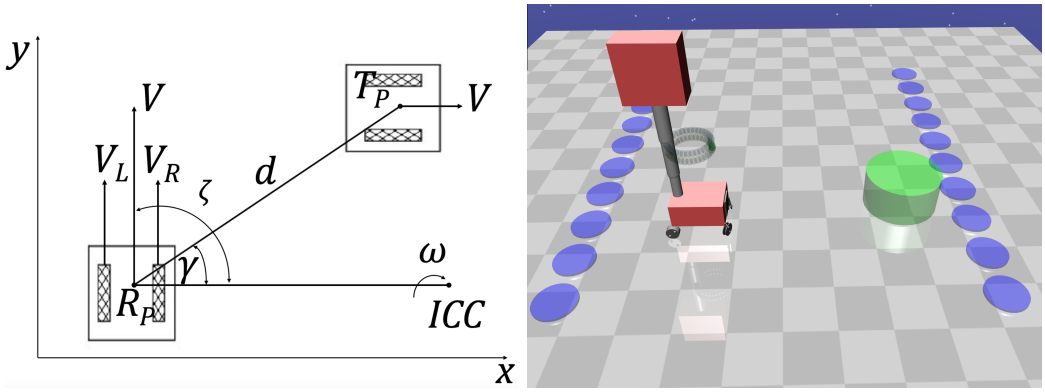
attached to the base and the last one to the basket; the boom components are linked with each other through sliding joints, which allow the movement (upon actuation) along the z-axis. In particular, for each sliding joint the *dumping* and *stiffness* values are tuned to obtain the desired vibration settling time. In addition, a null *springref* value is considered to specify the angle along which the joint spring reaches the equilibrium. Table 8.1 summarizes the physical properties of the simulated DVML platform, while Table 8.2 reports the different joints and the configuration of the MuJoCo parameters to realize each desired movement.

It is important to notice that the actuation of the boom during navigation is forbidden by the regulations of such machine. For this reason, the height of the basket becomes a fixed parameter for each test and is, therefore, kept out from the set of control inputs. Finally, a LiDAR sensor has been employed onboard of the robot for the obstacle detection.

8.4.1 Autonomous Navigation of a DVML

As mentioned before, DVML typically employ a DD mechanism which consists of: 1) two independently motorized wheels, mounted on a common axis; 2) two castor wheels, used to ensure the vehicle static stability. By applying different velocities on the wheels, the robot can follow trajectories coherently with its kinematics constraints.

The autonomous navigation problem regards the pursuit of a set of target coordinates $T_P = (x_T, y_T)$ from the current robot's position $R_P = (x_R, y_R)$. In other words, the *controller* must provide a set of control inputs to drive the robot from R_P to T_P. A further requirement is represented by the vibrations of the boom: the controller of a DVML must not only bring the robot to the desired position, but must minimize vibrations throughout the entirety of trajectory.



(a) Model notation (b) Test environment and robot model

Figure 8.4.1

Figure 8.4.1(a) shows the typical scenario and the involved variables in the problem of controlling the position of a DD mobile robot, where d represents the linear distance between the current position and the target one.

This chapter focuses on equipping a DVML with autonomous driving capabilities. In particular, it analyses the performance of the controller in logistic and industrial environments, typically composed of two main types of working surroundings: large working areas (which will be represented with areas without walls for simplicity) in which shelves, boxes and humans can operate, and hallways and corridors, with much less obstacles in the way. The environment in which the robot will move is implemented in *OpenAI* SafetyGym (SG) [[185], [189]], an environment builder which leverages MuJoCo as physics simulator and is compliant to *OpenAI Gym*, one of the most popular frameworks for building RL environments. SG has been chosen due to its flexibility in the creation of new environments, which can be composed of a wide range of physics items, targets, and – more importantly – safety constraints. Moreover, a relevant feature of SG is the possibility of randomizing the environment's layout at the beginning of each episode. This allows to prevent RL agents from learning trivial solutions to a particular setting of the environment and, instead, pushes the agent to learn more general behaviors with respect to the given task.

In each simulation, a green cylinder (see Figure 8.4.1(b)) at a fixed position visualizes the goal area. Obstacles are represented by *hazards* (blue circles in Figure 8.4.1(b)). Driving through obstacles is considered as a constraint violation.

8.5 Environment Setting

In order to allow a fair comparison among all RL algorithms, the observation space, the action space, the reward function and the cost function are fixed for all experiments. The **Observation Space** is defined as the concatenation of: the linear distance from the goal, defined as the exponential of the negative distance $d = e^{-||T_P - R_P||}$; the LiDAR information that only detects obstacles in the scene (the information is represented as thirty two distances along thirty two different angles scanning the area around the robot); the position of the goal, arranged as a LiDAR reading. The **Action Space** is composed of the velocities of each wheel, each in the range $[-1, 1]$. The **Reward Function** is defined as the sum of two main components: a utility function, represented by the instantaneous variation of the distance Δd from the goal, and a cost function. The cost function is composed of two terms: a penalization for any vibration of the boom ψ, computed as $\psi = ||P_{base} - P_{basket}||$ with P_{base} and P_{basket} being respectively the base position and the basket position; a constraint violation term η, proportional to the distance of the robot from the center of the hazard. The latter is added only when the robot is trespassing the hazard (i.e., when the robot-hazard distance is less than the hazard's size). Finally, a bonus β is added only when the linear distance is below a given threshold (tolerance) ρ: $||T_P - R_P|| < \rho$. Note that both β and ρ are hyperparameters to be tuned. The resulting reward function is reported below:

$$r_t = \begin{cases} -\Delta d - \psi - \eta, & \text{if } d \geq \rho \\ -\Delta d - \psi - \eta + \beta, & \text{if } d < \rho \end{cases} \tag{8.3}$$

where

$$\eta = \begin{cases} 0, & \text{if } ||H_P - R_P|| \geq H_s \\ ||H_P - R_P||, & \text{if } ||H_P - R_P|| \leq H_s \end{cases} \tag{8.4}$$

and H_P being the hazard's position and H_s the hazard's size.

8.6 Results

In order to obtain a good representation of the state of art performance the following algorithms are included:

1. TRPO, and its Lagrangian version, called TRPO Lag;
2. PPO, and its Lagrangian version, called PPO Lag;
3. PPO with ISSA, called PPO ISSA;

4. Lagrangian version of DDPG, called DDPG Lag;

5. CPO;

6. CVPO.

Note that the CVPO and PPO ISSA implementations faithfully follow the one proposed by their respective authors, while the remaining algorithms are implemented following the implementation proposed by [185], except DDPG Lagrangian whose implementation follows [169].

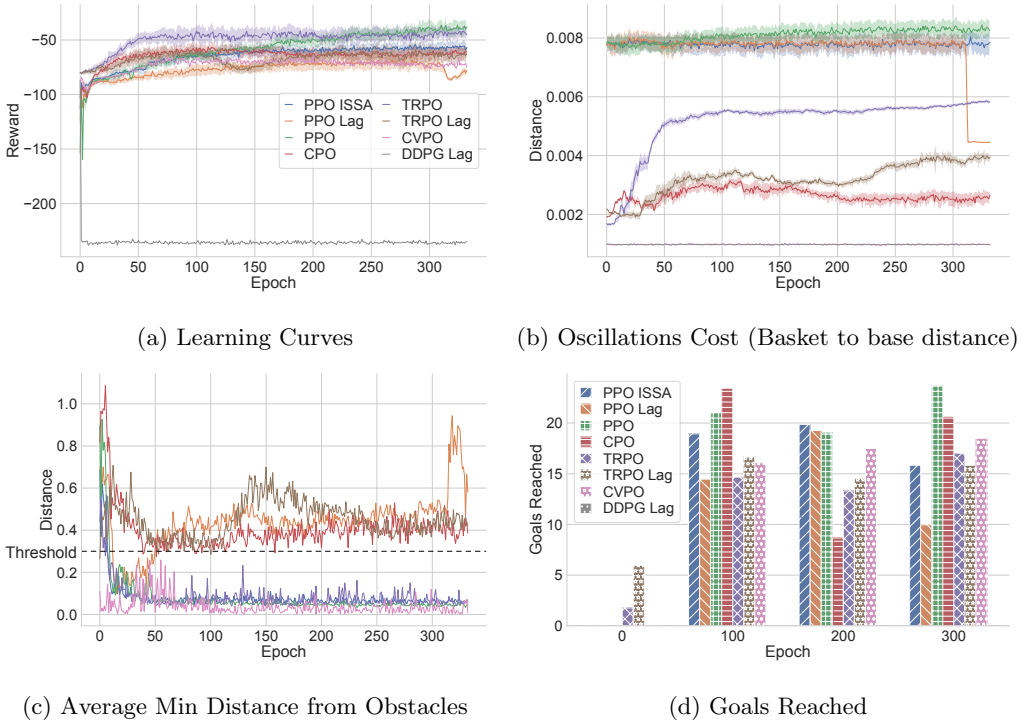
Each algorithm has been trained on an environment in which, at each episode, three obstacles and the goal are positioned randomly. The number of obstacles is carefully fixed in order to study the generalization capabilities during tests. For each algorithm, several configurations have been tested. For brevity, only the best ones are shown. During training, the simulation is not interrupted neither if a violation happens, nor if the agent reaches the goal. In the latter case, a new goal is generated.

In order to carry out a fair comparison, all algorithms are trained for the same number of epochs and employ the same neural network structure (two hidden layers, 256 neurons each). The number of epochs is fixed to 333 for all algorithms, however, the number of interaction steps is set to 10^7 for all algorithms, except CVPO and DDPG Lag, whose number of interaction steps is limited to $333 \cdot 10^3$. This difference is due to the different implementations adopted. In fact, each algorithm implementation is optimized with respect to such numbers of interaction steps and tests with different values have lead to worse performance. For each algorithm, multiple hyper-parameters configurations have been tested. For brevity, only the configurations associated with best performance are retained.

To test generalization capabilities of the agents, tests will be carried out in two different settings: 1) a first environment with twenty hazards to simulate a hallway, typical of industrial/logistics scenarios 8.4.1(b); 2) a second setting adopts different DVML physical parameters, in particular higher basket's weight and extended boom.

8.6.1 Overall results

In Figure 8.6.1, the overall performance during training are reported. In particular, observing Figure 8.6.1(a), all agents show a good progression in terms of reward function. In fact, it would seem that PPO and TRPO are the best performing algorithms. Given the particular application, a key aspect to be taken into account is the intensity of oscillations. During training all algorithms produce very distinct behaviours (see Figure 8.6.1(b)): while PPO and PPO ISSA seem to ignore such signal (they produced oscillations in the same range even after lots of updates), TRPO and TRPO Lagrangian worsen their performance as training goes on. On the other hand, CPO and



(a) Learning Curves

(b) Oscillations Cost (Basket to base distance)

(c) Average Min Distance from Obstacles

(d) Goals Reached

Figure 8.6.1: Overall Training Performance

PPO Lagrangian show an increase in vibrations' intensity during the first few epochs of training, but stabilize to much lower values towards the end. This is the behaviour to be expected: initial movements will be ineffective, producing neglectable oscillations; as the agent learns to drive the robot oscillations will increase due to suboptimal accelerations; later on, the agent will be able to optimize its driving abilities with respect to oscillations. A peculiar trend (or absence of) is shown by CVPO, which seems to produce very low values of vibrations throughout the entirety of its training.

In Figure 8.6.1(c) and Figure 8.6.1(d) are presented respectively the minimum distance from obstacles and the number of goals reached as the policy is updated. In particular, observing Figure 8.6.1(d), almost all algorithms seem to learn to reach the goal (note that DDPG Lag never reached the goal and therefore its value is zero), however, this is not enough: as shown in Figure 8.6.1(c), only CPO, PPO Lag and TRPO Lag are able to satisfy the constraints. In particular, CPO is the algorithm that manages to optimize both constraint satisfaction and navigation to the goal. Note that, while the minimum distance from obstacles' signal numerically dominates the oscillation cost one, the former is only applied when the constraints are violated, without clouding the oscillation signal in the remaining time. An overall analysis suggests that CPO represents the

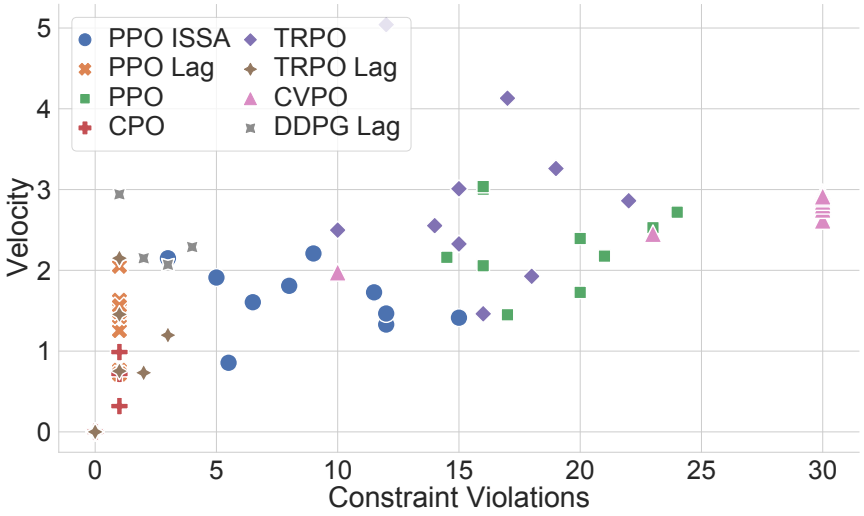


Figure 8.6.2: Average velocity when constraint violations happen, during Tests

best solution, since it is able to drive to the objective while minimizing the oscillations and satisfying constraints, at least in the final stages of training. In fact, not even CPO is able to guarantee constraint satisfaction throughout the training.

8.6.2 Detailed analysis

Figure 8.6.2 aims at providing an analysis of the severity of the constraint violations that occurred while testing each trained policy in an hallway-like environment. In particular, the number of constraint violations and the average velocity are compared: the best algorithm should be able to avoid constraint violations. If violations happen, it is preferable to have very low velocities. Therefore, the best algorithms will be found in the bottom left corner of such figure, whereas the top right area corresponds to algorithms with the poorest performance. By observing this figure, the best performing algorithm is CPO. In particular, it manages to satisfy the constraints in most cases and, when violations occur, it is driving the vehicle at a very low speed. Concerning TRPO Lagrangian and PPO Lagrangian, while they are able to reach the goal a sufficient number of times, they also lead the robot towards numerous constraint violations, some of them at high speeds, too. Note that Figure 8.6.2 is limited to 30 violations for improved visibility, however, without such limitation, one could observe that CVPO reaches hundreds of violations during test at roughly $3\frac{m}{s}$, which is unacceptable.

To further analyze constraint satisfaction, in Figure 8.6.3 is reported the cumulative distribution function of the minimum distance of the robot from the obstacles during tests. It is also reported

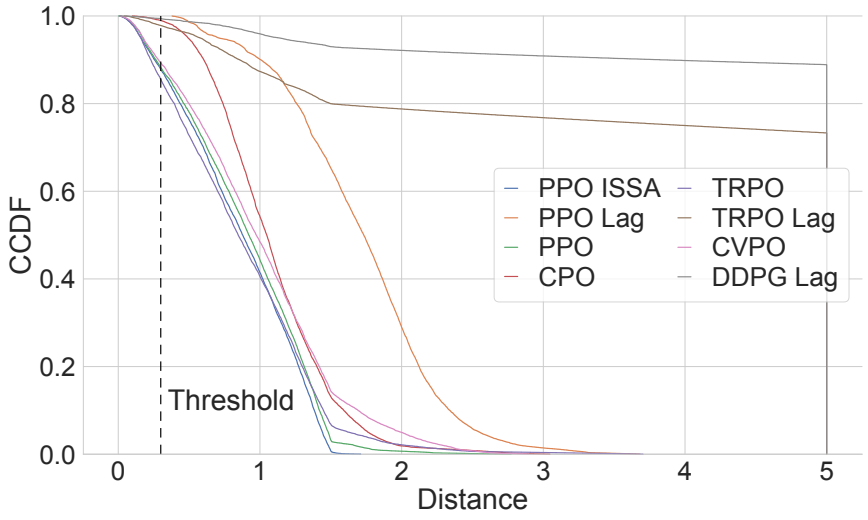


Figure 8.6.3: CCDF of Minimum Distance from Obstacles during Tests

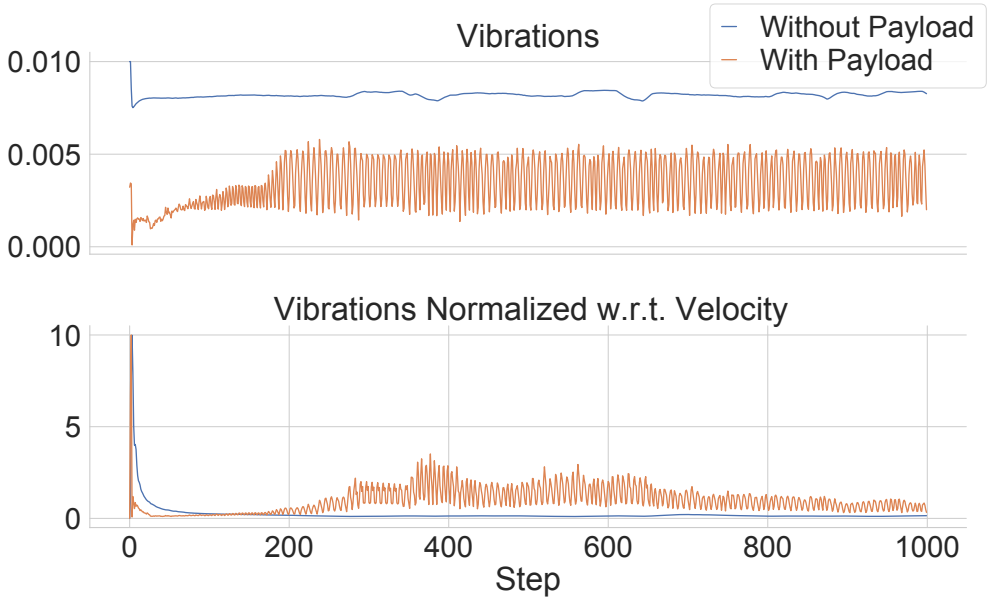


Figure 8.6.4: Vibration evolution of CPO agent

the threshold under which a constraint violation is detected. This figure mainly shows two aspects: 1) none of the agents are able to always satisfy the safety constraints; 2) CPO, PPO Lag, TRPO and TRPO Lag show the best performance. However only CPO and PPO Lag stay actually close to the goal: the remaining agents drive, in the majority of cases, the robot very far from the obstacles (and, hence, also from the goal) exhibiting a very dangerous and unacceptable behaviour.

In Figure 8.6.4 is considered the CPO agent, which is the one associated with best performance. In order to further investigate the generalization capabilities of the agent with respect to the physical system parameters, the policies are tested by increasing the height of the boom of 3 meters and adding 100kg to the basket's payload, in order to simulate the weight of a human operator and other tools. Figure 8.6.4 shows that the vibrations' intensity with and without payload are comparable with each other (and in both cases neglectable). Since trajectories produced in different conditions will differ, also the vibrations normalized with respect to the current velocity, depicted in Figure 8.6.2, are analysed. In fact, Since the intensity is comparable but a higher payload will make the robot move slower, the normalized vibrations will have a higher value. Nonetheless, values are still comparable with each other, highlighting how such model is able to generalize its driving skills to robots with different physical parameters.

8.6.3 Discussion

The results show that the best performance are obtained, as expected, by policies trained with Safe RL algorithms. In fact, even if some of the RL algorithms manage to reach several times the goal position, they are unable to satisfy the constraints. The best performing algorithm has been tested on different settings of the physical system and has shown good robustness to the variation. However, even the best algorithm is unable to guarantee the constraint satisfaction during training. The best performance trade-off between controlling the robot's position and the satisfying the safety constraints, in terms of both vibrations minimization and of constraint satisfaction, are obtained by the CPO algorithm. The experiments carried out clearly exhibit a gap between Safe RL methodologies proposed by the state of art and their employment on real industrial vehicles. In fact, while in proof-of-concept environments such algorithms show good performance, on real industrial use cases, such as the one proposed in this chapter, none of the considered methodologies are capable of guaranteeing constraints satisfaction neither throughout training nor during tests. Moreover, such algorithms can exhibit an unacceptable dynamic behavior, i.e. can impact hazards at high speeds. In order to allow Safe RL to actually be employed on real use cases, there is the need to prove such methodologies on real scenarios that can highlight criticalities.

Safe Reinforcement Learning and Constraint Violations

It is important to highlight that this chapter focused on *model-free* SRL algorithms. Being model-free, these algorithms cannot leverage any explicit information (or mathematical formulation) of the constraints. In fact, all approaches try to synthesize a control policy by computing updates from rewards and costs signals. However, the information related to such signals is strictly related to how the rewards and costs functions are designed. To effectively avoid constraint violations, these algorithms require a sufficiently high amount these signals, i.e. more interactions with the environment. This means that, especially in the early stages where the agent is usually pseudo-randomly initialized, ensuring constraint satisfaction for complex dynamical systems becomes a non-trivial problem.

Other approaches, which can be included in a future comparison, leverage barrier functions [190–192] to prevent interactions with the environment that would lead to constraint violations.

Another kind of SRL approaches are *model-based* algorithms [193, 194], which usually leverage a model of the system, a model of part of the system, or an estimate of the model of the system to prevent constraint violations. However, these algorithms are typically less employed due to the need for a model, which may not always be available. Moreover, if a model is available, classical and modern controller may solve the same problem with final improved performance.

8.7 Concluding Remarks

The rapid growth of Industry 4.0 is leading towards industrial, logistic, and smart agriculture scenarios in which CPS are employed to carry out complex tasks in changing environments in which humans can navigate too. In these scenarios, safety not only is a concern, but a hard constraint which must be satisfied at all time. This chapter considered the task of equipping a DVML with autonomous driving capabilities with a Safe RL approach. The main state of art Safe RL algorithms are trained on a real industrial scenario in which DMVLs are required to navigate in a complex environment. The best performing algorithm is tested on very different physical parameters with respect to the values observed during training and shows good generalization capabilities, yet it is still not able to satisfy the safety constraints in every case. While Safe RL brings the great advantage of synthesizing a control policy that allows neglecting the underlying model and requires hand-made hyperparameters fine-tuning, validating these methodologies in simple scenarios can lead to approaches which hardly generalize to complex scenarios, hence creating a gap with research and industrial applications, keeping such methodologies from being actually employed.

Chapter 9

Conclusions and Future Research Directions

This thesis is dedicated to the study of robotics and machine learning, presenting deep learning approaches in both perception and control fields.

The first part of this thesis deals with robotics perception, proposing novel approaches and datasets capturing real world use cases.

The first chapter of this part proposes Point2Depth, a cross-modal contrastive learning approach to train an autoencoder to map mmWave point clouds into depth images. This work aims at taking advantage of both modalities: mmWave radars are cheap sensors, robust to environmental conditions (like fog, smoke, obstacles...) and are being increasingly used in industrial applications. However, the point cloud produced by these sensors is often sparse and the points between frames are sporadic, meaning that many points appear and disappear frame by frame, mostly due to noise. On the other hand, depth images are readable data which can be directly used to map the environment. The trained cGAN will consistently produce depth images frame by frame, using an out-of-the-box mmWave sensor, effectively taking advantage of the two type of data. The approach has been validated on simple, complex and completely unseen environment, showing its reliability and generalisation capabilities.
An interesting evolution of this work could see the model leveraging frame-by-frame information, including recurrent neural networks in its architecture. In fact, exploiting the temporal sequence of multiple point cloud frames my advance the performance of the model, making it more accurate and more robust to sporadic points.

The second chapter delves into mmWave radars, presenting MilliNoise, a 12 million point cloud dataset for indoor scenarios. Each point is individually and accurately labelled. The label accuracy

is only related to the accuracy of the motion tracking system, which is used as ground truth of the pose of the robot in the environment and achieves sub-millimeter accuracy. Along with accurate labels, this dataset comes with the distance to the closest object in the scene for each point, enabling regression techniques for mmWave point clouds denoising tasks.

Following this work, the third chapter of the first part proposes an architecture to denoise such peculiar point clouds. After benchmarking the state of the art techniques and highlighting how these are associated to shortcomings when dealing with noisy and sparse point clouds, the chapter presents GT-MilliNoise, a transformer architecture based on graph neural networks which elaborates mmWave point clouds both on a time scale and from a geometrical point of view. The model, validated on the MilliNoise, shows improved performance with respect to the state of the art, both from a qualitative and a quantitative point of view. The ablation study conducted reveals how the same sporadic points that hinder the performance of the state of the art, become a key resource of the proposed method, which leverages such information to better detect noise. Note that, while it improves performance with respect to the state of the art, the proposed model is far from perfect. Despite its good performance, there is still margin for improvements. Another interesting future development could be reducing the size of such model, enabling its use on embedded devices.

The final chapter of this part focuses on UAVs applications. In particular, it proposes APEIRON, a multi-modal dataset at the intersection of perception, telecommunication and control of drones in outdoor scenarios. In particular, the dataset collects 1) several perception data, including high-resolution event-based camera, an RGB-D camera, gyroscope, accelerometer, and GPS data; 2) live telecommunication data, like bandwidth maps and network traces; and 3) low level control data of the drone. This dataset is proposed as one of the first steps to enable machine learning models in approaches that consider perception and control of robots altogether with telecommunication data. As a future research, it would be interesting to leverage a model to predict network traces given the position of the robot, or optimizing swarm trajectories by taking into account of bandwidth maps as well.

The second part deals with intelligent control of industrial robots, proposing tools to advance Reinforcement Learning performance and applying Safe Reinforcement Learning algorithms to industrial robots subject to safety regulations and laws.

In particular, the first chapter of the second part deals with the pose regulation of a *four wheel steering four wheel driving* robot, known for its peculiar dynamics and kinematics, which makes it hard to design a controller that actually takes advantage of its *pseudo-omnidirectionality*. As the benchmark of state of the art Reinforcement Learning algorithms reveals, these peculiarities actually hinder the learning process too. For these reasons, two tools are proposed: an *Episodic Noise*, which

enhances the exploration and enables complex behaviours to emerge already in the first training epochs; and the DM, a tool that applies curriculum learning by calibrating the difficulty of the target poses based on the current model capability. The two tools effectively improve performance, enabling the RL algorithm to converge to an optimal solution in just a few hundred training epochs. An interesting future research direction would be to validate the proposed tools on a wider range of applications and environments and study how the two tools affect the learning process in these new contexts.

The final chapter of this thesis deals with the autonomous navigation of a DVML, an industrial wheeled mobile robot subject to safety regulations and laws, due to its weight and dynamics. In order to train a model while dealing with safety constraints, whose main aim is to prevent crashes and tip-overs, Safe Reinforcement Learning algorithms have been employed. In particular, a benchmark of the state-of-the-art SRL algorithms has been carried out. The best performing algorithm has also been tested by changing the physical parameters of the robot, showing good generalization capabilities. Nevertheless, it is still not able to satisfy the safety constraints: there are still particular cases and trajectories that cause the robot to violate the constraints. This study highlighted how SRL algorithms still have issues and shortcomings when it comes to real-world applications. The study shows how validating such algorithms on proof-of-concept scenarios is often not enough to highlight the limits of the approaches, hence hindering a wider adoption of these solutions in industrial applications. An interesting research direction could be to use the tools proposed in the previous chapter to enhance the performance of the SRL algorithms and validate the performance on a real industrial robot.

Combining the studies carried out during this thesis, it would be interesting to design approaches that combine perception data from a swarm of robots to enhance the perception of the whole group. Another interesting research direction could also deploy DRL algorithms to control a swarm of robots while taking into account current bandwidth measurements, in order to adjust the trajectories to avoid loss of communication.

Bibliography

[1] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, 1943.

[2] A. Newell and H. A. Simon, "Report on a general problem-solving program," tech. rep., Carnegie Institute of Technology, 1959.

[3] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65 6, pp. 386–408, 1958.

[4] H. D. Block, "The perceptron: A model for brain functioning. i," *Rev. Mod. Phys.*, vol. 34, pp. 123–135, Jan 1962.

[5] M. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 09 2017.

[6] A. Bryson and Y. Ho, *Applied Optimal Control: Optimization, Estimation, and Control*. Blaisdell book in the pure and applied sciences, Blaisdell Publishing Company, 1969.

[7] H. Andreasson, G. Grisetti, T. Stoyanov, and A. Pretto, "Sensors for mobile robots," in *Encyclopedia of Robotics* (M. H. Ang, O. Khatib, and B. Siciliano, eds.), pp. 1–22, Springer Berlin Heidelberg, 2020.

[8] R. E. Kalman, *A New Approach to Linear Filtering and Prediction Problems*, vol. 82, pp. 35–45. Wiley, 03 1960.

[9] J. Rawlings, D. Mayne, and M. Diehl, "Model predictive control: Theory, computation, and design," 01 2017.

[10] T. Mitchell, *Machine Learning*. McGraw-Hill International Editions, McGraw-Hill, 1997.

[11] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[12] H. Hotelling, "Analysis of a complex of statistical variables into principal components.," *Journal of Educational Psychology*, vol. 24, pp. 498–520, 1933.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[14] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, pp. 1771–1800, 08 2002.

[15] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1, pp. 539–546, IEEE, 2005.

[16] G. E. Hinton and R. Zemel, "Autoencoders, minimum description length and helmholtz free energy," *Advances in neural information processing systems*, vol. 6, 1993.

[17] C. Chen, S. Rosa, C. X. Lu, B. Wang, N. Trigoni, and A. Markham, "Learning selective sensor fusion for states estimation," 2022.

[18] C. Wang, C. Ma, M. Zhu, and X. Yang, "Pointaugmenting: Cross-modal augmentation for 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11794–11803, June 2021.

[19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[21] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Jozefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. Pinto, J. Raiman, T. Salimans, J. Schlatter, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," 2019.

[22] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, (New York, NY, USA), p. 197–210, Association for Computing Machinery, 2017.

[23] V. Martín, J. Cabrera, and N. García, "Design, optimization and evaluation of a q-learning http adaptive streaming client," *IEEE Transactions on Consumer Electronics*, vol. 62, no. 4, pp. 380–388, 2016.

[24] B. O. Turkkan, T. Dai, A. Raman, T. Kosar, C. Chen, M. F. Bulut, J. Zola, and D. Sow, "Greenabr: energy-aware adaptive bitrate streaming with deep reinforcement learning," in *Proceedings of the 13th ACM Multimedia Systems Conference*, MMSys '22, (New York, NY, USA), p. 150–163, Association for Computing Machinery, 2022.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[26] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," *arXiv*, vol. arXiv:1801.08757, 2018.

[27] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.

[28] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[29] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, pp. 729–734 vol. 2, 2005.

[30] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 1025–1035, Curran Associates Inc., 2017.

[31] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[32] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (A. Moschitti, B. Pang, and W. Daelemans, eds.), (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.

[33] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[34] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[35] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, Apr. 2018.

[36] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, pp. 1889–1897, PMLR, 2015.

[37] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, pp. 1861–1870, PMLR, 2018.

[38] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. of ICLR*, 2015.

[39] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[40] P. Zhao, C. X. Lu, J. Wang, C. Chen, W. Wang, N. Trigoni, and A. Markham, "mid: Tracking and identifying people with millimeter wave radar," in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 33–40, IEEE, 2019.

[41] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proc. of IEEE conference on CVPR*, pp. 652–660, 2017.

[42] J.-T. Huang, C.-L. Lu, P.-K. Chang, C.-I. Huang, C.-C. Hsu, Z. L. Ewe, P.-J. Huang, and H.-C. Wang, "Cross-modal contrastive learning of representations for navigation using lightweight, low-cost millimeter wave radar for adverse environmental conditions," *IEEE Robotics and Automation Letters*, vol. 6, p. 3333–3340, Apr. 2021.

[43] S. M. S. S. Basha, S. Sridhar, S. Kaushik, and H. Kumar, "Millinet: Applied deep learning technique for millimeter-wave based object detection and classification," *IETE Journal of Research*, vol. 0, no. 0, pp. 1–7, 2022.

[44] Y. Sun, Z. Huang, H. Zhang, Z. Cao, and D. Xu, "3DRIMR: 3D reconstruction and imaging via mmWave radar based on deep learning," in *Proc of. IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pp. 1–8, 2021.

[45] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *Advances in neural information processing systems*, vol. 33, pp. 18661–18673, 2020.

[46] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[47] W. Brescia, G. Roberto, V. A. Racanelli, S. Mascolo, and L. D. Cicco, "Point2depth: a gan-based contrastive learning approach for mmwave point clouds to depth images transformation," in *2023 31st Mediterranean Conference on Control and Automation (MED)*, pp. 529–534, 2023.

[48] O. Henaff, "Data-efficient image recognition with contrastive predictive coding," in *ICML*, pp. 4182–4192, PMLR, 2020.

[49] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification.," *Journal of machine learning research*, vol. 10, no. 2, 2009.

[50] M. Laskin, A. Srinivas, and P. Abbeel, "Curl: Contrastive unsupervised representations for reinforcement learning," in *ICML*, pp. 5639–5650, PMLR, 2020.

[51] 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*, vol. 63, no. 11, pp. 139–144, 2020.

[52] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proc. of IEEE conference on CVPR*, pp. 1125–1134, 2017.

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[54] Z. Chen, V. Badrinarayanan, G. Drozdov, and A. Rabinovich, "Estimating depth from RGB and sparse sensing," in *Proc. of European Conference on Computer Vision (ECCV)*, pp. 167–182, 2018.

[55] J. Hu, M. Ozay, Y. Zhang, and T. Okatani, "Revisiting single image depth estimation: Toward higher resolution maps with accurate object boundaries," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1043–1051, 2019.

[56] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[57] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," 2016.

[58] H. Yin, X. Xu, Y. Wang, and R. Xiong, "Radar-to-lidar: Heterogeneous place recognition via joint learning," *Frontiers in Robotics and AI*, vol. 8, 2021.

[59] F. Kraus, N. Scheiner, W. Ritter, and K. Dietmayer, "The Radar Ghost Dataset – An Evaluation of Ghost Objects in Automotive Radar Data," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8570–8577, 2021.

[60] V. Sanchez and A. Zakhor, "Planar 3d modeling of building interiors from point cloud data," in *Proc. of the 19th IEEE International Conference on Image Processing*, pp. 1777–1780, 2012.

[61] G. Zhang, X. Geng, and Y.-J. Lin, "Comprehensive mpoint: A method for 3d point cloud generation of human bodies utilizing fmcw mimo mm-wave radar," *Sensors*, vol. 21, no. 19, 2021.

[62] Y. Almalioglu, M. Turan, C. X. Lu, N. Trigoni, and A. Markham, "Milli-rio: Ego-motion estimation with low-cost millimetre-wave radar," *IEEE Sensors Journal*, vol. 21, no. 3, pp. 3314–3323, 2021.

[63] S. Palipana, D. Salami, L. A. Leiva, and S. Sigg, "Pantomime: Mid-air gesture recognition with sparse millimeter-wave radar point clouds," *Proc. of the ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 5, mar 2021.

[64] B. Han, Y. Liu, and F. Qian, "ViVo: visibility-aware mobile volumetric video streaming," in *Proc. of the 26th ACM Annual International Conference on Mobile Computing and Networking*, MobiCom '20, 2020.

[65] I. Viola and P. Cesar, "Chapter 15 - volumetric video streaming: Current approaches and implementations," in *Immersive Video Technologies* (G. Valenzise, M. Alain, E. Zerman, and C. Ozcinar, eds.), pp. 425–443, Academic Press, 2023.

[66] C. X. Lu, S. Rosa, P. Zhao, B. Wang, C. Chen, J. A. Stankovic, N. Trigoni, and A. Markham, "See through smoke: robust indoor mapping with low-cost mmwave radar," in *Proc. of the 18th International Conference on Mobile Systems, Applications, and Services*, MobiSys '20, p. 14–27, 2020.

[67] Y. Cheng, H. Xu, and Y. Liu, "Robust small object detection on the water surface through fusion of camera and millimeter wave radar," in *Proc. of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15243–15252, 2021.

[68] A. Venon, Y. Dupuis, P. Vasseur, and P. Merriaux, "Millimeter wave fmcw radars for perception, recognition and localization in automotive applications: A survey," *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 3, pp. 533–555, 2022.

[69] X. Li, X. Wang, Q. Yang, and S. Fu, "Signal processing for tdm mimo fmcw millimeter-wave radar sensors," *IEEE Access*, vol. 9, pp. 167959–167971, 2021.

[70] Y. Li, Y. Liu, Y. Wang, Y. Lin, and W. Shen, "The millimeter-wave radar slam assisted by the rcs feature of the target and imu," *Sensors*, vol. 20, no. 18, 2020.

[71] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proc. of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11621–11631, 2020.

[72] M. Meyer and G. Kuschk, "Automotive radar dataset for deep learning based 3d object detection," in *Proc. of 16th European Radar Conference (EuRAD)*, pp. 129–132, 2019.

[73] A. Ouaknine, A. Newson, J. Rebut, F. Tupin, and P. Pérez, "Carrada dataset: Camera and automotive radar with range- angle- doppler annotations," in *Proc. of the 25th International Conference on Pattern Recognition (ICPR)*, pp. 5068–5075, 2021.

[74] O. Schumann, M. Hahn, N. Scheiner, F. Weishaupt, J. F. Tilly, J. Dickmann, and C. Wöhler, "Radarscenes: A real-world radar point cloud data set for automotive applications," in *Proc. of the IEEE 24th International Conference on Information Fusion (FUSION)*, pp. 1–8, 2021.

[75] M. Chamseddine, J. Rambach, D. Stricker, and O. Wasenmuller, "Ghost target detection in 3d radar data using point cloud based deep neural network," in *Proc. of the 25th International Conference on Pattern Recognition (ICPR)*, pp. 10398–10403, 2021.

[76] P. Li, K. Cai, M. R. U. Saputra, Z. Dai, and C. X. Lu, "OdomBeyondVision: An Indoor Multi-modal Multi-platform Odometry Dataset Beyond the Visible Spectrum," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3845–3850, 2022.

[77] Y.-H. Wu, H.-C. Chiang, S. Shirmohammadi, and C.-H. Hsu, "A Dataset of Food Intake Activities Using Sensors with Heterogeneous Privacy Sensitivity Levels," in *Proc. of the 14th Conference on ACM Multimedia Systems*, MMSys '23, p. 416–422, 2023.

[78] A. Sengupta, F. Jin, R. Zhang, and S. Cao, "mm-Pose: Real-Time Human Skeletal Posture Estimation Using mmWave Radars and CNNs," *IEEE Sensors Journal*, vol. 20, no. 17, pp. 10032–10044, 2020.

[79] H. Cui, S. Zhong, J. Wu, Z. Shen, N. Dahnoun, and Y. Zhao, "MiliPoint: A Point Cloud Dataset for mmWave Radar," 2023.

[80] A. D. Singh, S. S. Sandha, L. Garcia, and M. Srivastava, "Radhar: Human activity recognition from point clouds generated through a millimeter-wave radar," in *Proc. of the 3rd ACM Workshop on Millimeter-Wave Networks and Sensing Systems*, mmNets'19, p. 51–56, 2019.

[81] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, 2012.

[82] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proc. of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2446–2454, 2020.

[83] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, J. Gall, and C. Stachniss, "Towards 3d lidar-based semantic scene understanding of 3d point cloud sequences: The semantickitti dataset," *The International Journal of Robotics Research*, vol. 40, no. 8-9, pp. 959–967, 2021.

[84] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *Proc. of the IEEE International Conference on Image Processing (ICIP)*, pp. 3464–3468, 2016.

[85] J. Kopp, D. Kellner, A. Piroli, and K. Dietmayer, "Tackling Clutter in Radar Data - Label Generation and Detection Using PointNet++," in *Proc. of ICRA*, pp. 1493–1499, 2023.

[86] T. Griebel, D. Authaler, M. Horn, M. Henning, M. Buchholz, and K. Dietmayer, "Anomaly Detection in Radar Data Using PointNets," in *Proc. of the IEEE International Intelligent Transportation Systems Conference (ITSC)*, p. 2667–2673, 2021.

[87] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proc. of ICRA*, vol. 2, pp. 1322–1328, 1999.

[88] M. R. Anderberg, *Cluster Analysis for Applications*. Probability and Mathematical Statistics: A Series of Monographs and Textbooks, Academic Press, 1973.

[89] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[90] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions on Graphics*, 2019.

[91] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[92] K. Mazur and V. Lempitsky, "Cloud transformers: A universal approach to point cloud processing tasks," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10695–10704, 2021.

[93] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 16259–16268, October 2021.

[94] X. Wu, L. Jiang, P.-S. Wang, Z. Liu, X. Liu, Y. Qiao, W. Ouyang, T. He, and H. Zhao, "Point transformer v3: Simpler faster stronger," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4840–4851, June 2024.

[95] M. Chamseddine, J. Rambach, D. Stricker, and O. Wasenmuller, "Ghost target detection in 3d radar data using point cloud based deep neural network," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 10398–10403, 2021.

[96] Y. Jin, R. Prophet, A. Deligiannis, I. Weber, J.-C. Fuentes-Michel, and M. Vossiek, "Comparison of different approaches for identification of radar ghost detections in automotive scenarios," in *2021 IEEE Radar Conference (RadarConf21)*, pp. 1–6, 2021.

[97] F. Kraus, N. Scheiner, W. Ritter, and K. Dietmayer, "Using machine learning to detect ghost images in automotive radar," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–7, IEEE, 2020.

[98] W. Wang, R. Yu, Q. Huang, and U. Neumann, "Sgpn: Similarity group proposal network for 3d point cloud instance segmentation," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2569–2578, 2018.

[99] A. Cennamo, F. Kaestner, and A. Kummert, "Leveraging radar features to improve point clouds segmentation with neural networks," in *Proceedings of the 21st EANN (Engineering Applications of Neural Networks) 2020 Conference: Proceedings of the EANN 2020 21*, pp. 119–131, Springer, 2020.

[100] J. Liu, W. Xiong, L. Bai, Y. Xia, T. Huang, W. Ouyang, and B. Zhu, "Deep instance segmentation with automotive radar detection points," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 84–94, 2022.

[101] P. Veličković, "Message passing all the way up," *arXiv preprint arXiv:2202.11097*, 2022.

[102] T. Huang and Y. Liu, "3d point cloud geometry compression on deep learning," in *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, (New York, NY, USA), p. 890–898, Association for Computing Machinery, 2019.

[103] S. Cabello, P. Giannopoulos, C. Knauer, and G. Rote, "Matching point sets with respect to the earth mover's distance," *Computational Geometry*, vol. 39, no. 2, pp. 118–133, 2008.

[104] W. Brescia, P. Gomes, L. Toni, S. Mascolo, and L. De Cicco, "Millinoise: a millimeter-wave radar sparse point cloud dataset in indoor scenarios," in *Proceedings of the 15th ACM Multimedia Systems Conference*, MMSys '24, Association for Computing Machinery, 2024.

[105] J. Fan, L. Lei, S. Cai, G. Shen, P. Cao, and L. Zhang, "Area surveillance with low detection probability using uav swarms," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 2, pp. 1736–1752, 2024.

[106] B. Mishra, D. Garg, P. Narang, and V. Mishra, "Drone-surveillance for search and rescue in natural disaster," *Computer Communications*, vol. 156, pp. 1–10, 2020.

[107] S. Asadzadeh, W. J. de Oliveira, and C. R. de Souza Filho, "Uav-based remote sensing for the petroleum industry and environmental monitoring: State-of-the-art and perspectives," *Journal of Petroleum Science and Engineering*, vol. 208, p. 109633, 2022.

[108] K. R. Jensen-Nau, T. Hermans, and K. K. Leang, "Near-optimal area-coverage path planning of energy-constrained aerial robots with application in autonomous environmental monitoring," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 1453–1468, 2021.

[109] X. Wang, A. Chowdhery, and M. Chiang, "Networked drone cameras for sports streaming," in *Proc. of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 308–318, IEEE, 2017.

[110] M. A. Khan, E. Baccour, Z. Chkirbene, A. Erbad, R. Hamila, M. Hamdi, and M. Gabbouj, "A survey on mobile edge computing for video streaming: Opportunities and challenges," *IEEE Access*, vol. 10, pp. 120514–120550, 2022.

[111] J. Sabzehali, V. K. Shah, Q. Fan, B. Choudhury, L. Liu, and J. H. Reed, "Optimizing number, placement, and backhaul connectivity of multi-uav networks," *IEEE Internet of Things Journal*, vol. 9, no. 21, pp. 21548–21560, 2022.

[112] M. Gharibi, R. Boutaba, and S. L. Waslander, "Internet of drones," *IEEE Access*, vol. 4, pp. 1148–1162, 2016.

[113] S. J. Maeng, O. Ozdemir, I. Guvenc, M. L. Sichitiu, M. Mushi, and R. Dutta, "Lte i/q data set for uav propagation modeling, communication, and navigation research," *IEEE Communications Magazine*, vol. 61, no. 9, pp. 90–96, 2023.

[114] I. Medeiros, A. Boukerche, and E. Cerqueira, "Swarm-based and energy-aware unmanned aerial vehicle system for video delivery of mobile objects," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 1, pp. 766–779, 2021.

[115] H. Jiang, Y. Chang, L. Yang, X. Liu, and Y. He, "Cooperative Exploration of Heterogeneous UAVs in Mountainous Environments by Constructing Steady Communication," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, pp. 7249–7256, 2023.

[116] C. Song, B. Han, X. Ji, Y. Li, and J. Su, "AI-driven Multipath Transmission: Empowering UAV-based Live Streaming," *IEEE Network*, vol. in press, 2023.

[117] R. Makrigiorgis, C. Kyrkou, and P. Kolios, "How high can you detect? improved accuracy and efficiency at varying altitudes for aerial vehicle detection," in *Proc. of the International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 167–174, 2023.

[118] H.-J. Hsu and K.-T. Chen, "Droneface: An open dataset for drone research," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys'17, p. 187–192, 2017.

[119] J. Dupeyroux, R. Dinaux, N. Wessendorp, and G. De Croon, "A novel obstacle detection and avoidance dataset for drones," in *System Engineering for Constrained Embedded Systems*, DroneSE and RAPIDO, (New York, NY, USA), p. 8–13, Association for Computing Machinery, 2022.

[120] G. J. Martinez, G. Dubrovskiy, S. Zhu, A. Mohammed, H. Lin, J. N. Laneman, A. D. Striegel, R. V. Pragada, and D. R. Castor, "An Open, Real-World Dataset of Cellular

UAV Communication Properties," in *Proc. of the International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–6, 2021.

[121] N. Barone, W. Brescia, S. Mascolo, and L. De Cicco, "Apeiron: a multimodal drone dataset bridging perception and network data in outdoor environments," in *Proceedings of the 15th ACM Multimedia Systems Conference*, MMSys '24, (New York, NY, USA), p. 401–407, Association for Computing Machinery, 2024.

[122] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 154–180, 2022.

[123] J. Sae, R. Wirén, J. Kauppi, H.-L. Maattanen, J. Torsner, and M. Valkama, "Public LTE network measurements with drones in rural environment," in *Proc. of the IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pp. 1–5, IEEE, 2019.

[124] A. Dietsche, G. Cioffi, J. Hidalgo-Carrio, and D. Scaramuzza, "Powerline tracking with event cameras," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2021.

[125] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza, "Are we ready for autonomous drone racing? the uzh-fpv drone racing dataset," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6713–6719, 2019.

[126] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics and Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[127] D. Zha, K.-H. Lai, K. Zhou, and X. Hu, "Simplifying deep reinforcement learning via self-supervision," *arXiv preprint arXiv:2106.05526*, 2021.

[128] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

[129] H. Yang, V. Cocquempot, and B. Jiang, "Optimal fault-tolerant path-tracking control for 4WS4WD electric vehicles," *IEEE Transactions on intelligent transportation systems*, vol. 11, no. 1, pp. 237–243, 2009.

[130] X. Zhang, Y. Xie, L. Jiang, G. Li, J. Meng, and Y. Huang, "Fault-tolerant dynamic control of a four-wheel redundantly-actuated mobile robot," *IEEE Access*, vol. 7, pp. 157909–157921, 2019.

[131] J. Liao, Z. Chen, and B. Yao, "Performance-oriented coordinated adaptive robust control for four-wheel independently driven skid steer mobile robot," *IEEE Access*, vol. 5, pp. 19048–19057, 2017.

[132] W. Brescia, L. De Cicco, and S. Mascolo, "Sample-efficient reinforcement learning for pose regulation of a mobile robot," in *2022 11th International Conference on Control, Automation and Information Sciences (ICCAIS)*, pp. 42–47, 2022.

[133] S. Krishnan, B. Boroujerdian, W. Fu, A. Faust, and V. J. Reddi, "Air learning: a deep reinforcement learning gym for autonomous aerial robot visual navigation," *Machine Learning*, vol. 110, no. 9, pp. 2501–2540, 2021.

[134] C. P. Connette, C. Parlitz, M. Hagele, and A. Verl, "Singularity avoidance for over-actuated, pseudo-omnidirectional, wheeled mobile robots," in *Proc. of IEEE ICRA*, pp. 4124–4130, IEEE, 2009.

[135] C. P. Connette, A. Pott, M. Hagele, and A. Verl, "Control of an pseudo-omnidirectional, non-holonomic, mobile robot based on an icm representation in spherical coordinates," in *Proc. IEEE Conference on Decision and Control*, pp. 4976–4983, IEEE, 2008.

[136] C. Christian P., P. Andreas, H. Martin, and V. Alexander, "Addressing input saturation and kinematic constraints of overactuated undercarriages by predictive potential fields," *IROS*, 2010.

[137] S. Mohamed, C. Andrea, P. Robin, and F. Philippe, "Kinematic modeling and singularity treatment of steerable wheeled mobile robots with joint acceleration limits," *ICRA*, 2016.

[138] F. Scott, v. H. Herke, and M. David, "Addressing function approximation error in actor-critic methods," in *Proc. of ICML*, 2018.

[139] J.-B. Song and K.-S. Byun, "Steering control algorithm for efficient drive of a mobile robot with steerable omni-directional wheels," *Journal of mechanical science and technology*, vol. 23, no. 10, p. 2747, 2009.

[140] C. Florensa, D. Held, X. Geng, and P. Abbeel, "Automatic goal generation for reinforcement learning agents," in *Proc. of ICML*, pp. 1515–1528, PMLR, 2018.

[141] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5745–5753, 2019.

[142] D. Ferigo, S. Traversaro, G. Metta, and D. Pucci, "Gym-ignition: Reproducible robotic simulations for reinforcement learning," in *2020 IEEE/SICE International Symposium on System Integration (SII)*, pp. 885–890, 2020.

[143] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[144] S. Vaidya, P. Ambad, and S. Bhosle, "Industry 4.0 – a glimpse," *Procedia Manufacturing*, vol. 20, pp. 233–238, 2018.

[145] B. Dafflon, N. Moalla, and Y. Ouzrout, "The challenges, approaches, and used techniques of cps for manufacturing in industry 4.0: A literature review," *The International Journal of Advanced Manufacturing Technology*, vol. 113, pp. 2395–2412, 2021.

[146] R. Goel and P. Gupta, *Robotics and Industry 4.0*, ch. in: A Roadmap to Industry 4.0: Smart Production, Sharp Business and Sustainable Development, pp. 157–169. Springer International Publishing, 2020.

[147] A. H. El-Kady, S. Halim, M. M. El-Halwagi, and F. Khan, "Analysis of safety and security challenges and opportunities related to cyber-physical systems," *Process Safety and Environmental Protection*, vol. 173, pp. 384–413, 2023.

[148] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion planning and control for mobile robot navigation using machine learning: a survey," *Autonomous Robots*, vol. 46, no. 5, pp. 569–597, 2022.

[149] E. T. Maddalena, Y. Lian, and C. N. Jones, "Data-driven methods for building control — a review and promising future directions," *Control Engineering Practice*, vol. 95, p. 104211, 2020.

[150] Z. Wang and T. Hong, "Reinforcement learning for building controls: The opportunities and challenges," *Applied Energy*, vol. 269, p. 115036, 2020.

[151] M. L. Littman, "Reinforcement learning improves behaviour from evaluative feedback," *Nature*, vol. 521, no. 7553, pp. 445–451, 2015.

[152] H.-D. Tran, F. Cai, M. L. Diego, P. Musau, T. T. Johnson, and X. Koutsoukos, "Safety verification of cyber-physical systems with reinforcement learning control," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 105, pp. 1–22, 2019.

[153] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, no. 1, pp. 411–444, 2022.

[154] J. García, Fern, and o Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 42, pp. 1437–1480, 2015.

[155] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, Y. Yang, and A. Knoll, "A review of safe reinforcement learning: Methods, theory and applications," *arXiv*, vol. arXiv:2205.10330, 2023.

[156] A. Wachi and Y. Sui, "Safe reinforcement learning in constrained Markov decision processes," in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, pp. 9797–9806, PMLR, 2020.

[157] R. Singh, A. Gupta, and N. B. Shroff, "Learning in constrained markov decision processes," *IEEE Transactions on Control of Network Systems*, vol. 10, no. 1, pp. 441–453, 2022.

[158] W. Brescia, A. Maci, S. Mascolo, and L. De Cicco, "Safe reinforcement learning for autonomous navigation of a driveable vertical mast lift," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 9068–9073, 2023. 22nd IFAC World Congress.

[159] Y. Chow, O. Nachum, A. Faust, E. Duenez-Guzman, and M. Ghavamzadeh, "Lyapunov-based safe policy optimization for continuous control," *arXiv*, vol. arXiv:1901.10031, 2019.

[160] Y. Chow, O. Nachum, A. Faust, E. Dueñez-Guzman, and M. Ghavamzadeh, "Safe policy learning for continuous control," in *Proceedings of the 2020 Conference on Robot Learning*, vol. 155, pp. 801–821, PMLR, 2020.

[161] Y. S. Shao, C. Chen, S. Kousik, and R. Vasudevan, "Reachability-based trajectory safeguard (rts): A safe and fast reinforcement learning safety layer for continuous control," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3663–3670, 2021.

[162] L. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dubé, "Robot navigation in crowded environments using deep reinforcement learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5671–5677, IEEE, 2020.

[163] U. Patel, N. K. S. Kumar, A. J. Sathyamoorthy, and D. Manocha, "Dwa-rl: Dynamically feasible deep reinforcement learning policy for robot navigation among mobile obstacles," in *Proc. of ICRA*, pp. 6057–6063, IEEE, 2021.

[164] X. Zhang, Y. Peng, W. Pan, X. Xu, and H. Xie, "Barrier function-based safe reinforcement learning for formation control of mobile robots," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 5532–5538, IEEE, 2022.

[165] J. Thumm and M. Althoff, "Provably safe deep reinforcement learning for robotic manipulation in human environments," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 6344–6350, IEEE, 2022.

[166] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 22–31, PMLR, 2017.

[167] H. Ma, C. Liu, S. E. Li, S. Zheng, and J. Chen, "Joint synthesis of safety certificate and safe control policy using constrained reinforcement learning," in *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, vol. 168, pp. 97–109, PMLR, 2022.

[168] W. Zhao, T. He, and C. Liu, "Model-free safe control for zero-violation reinforcement learning," in *Proceedings of the 5th Conference on Robot Learning*, vol. 164, pp. 784–793, PMLR, 2021.

[169] Z. Liu, Z. Cen, V. Isenbaev, W. Liu, Z. S. Wu, B. Li, and D. Zhao, "Constrained variational policy optimization for safe reinforcement learning," in *Proceedings of the 39th International Conference on Machine Learning*, vol. 162, pp. 13644–13668, PMLR, 2022.

[170] Y. Zhang, Q. Vuong, and K. Ross, "First order constrained optimization in policy space," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 15338–15349, Curran Associates, Inc., 2020.

[171] T. Xu, Y. Liang, and G. Lan, "Crpo: A new approach for safe reinforcement learning with convergence guarantee," in *Proceedings of the 38th International Conference on Machine Learning*, vol. 139, pp. 11480–11491, PMLR, 2021.

[172] L. Zhang, L. Shen, L. Yang, S. Chen, X. Wang, B. Yuan, and D. Tao, "Penalized proximal policy optimization for safe reinforcement learning," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 3744–3750, International Joint Conferences on Artificial Intelligence Organization, 2022.

[173] L. Yang, J. Ji, J. Dai, L. Zhang, B. Zhou, P. Li, Y. Yang, and G. Pan, "Constrained update projection approach to safe policy optimization," in *Advances in Neural Information Processing Systems*, vol. 35, pp. 9111–9124, Curran Associates, Inc., 2022.

[174] A. Sootla, A. I. Cowen-Rivers, T. Jafferjee, Z. Wang, D. H. Mguni, J. Wang, and H. Ammar, "Saute RL: Almost surely safe reinforcement learning using state augmentation," in *Proceedings of the 39th International Conference on Machine Learning*, vol. 162, pp. 20423–20443, PMLR, 2022.

[175] A. Sootla, A. Cowen-Rivers, J. Wang, and H. Bou Ammar, "Enhancing safe exploration using safety state augmentation," in *Advances in Neural Information Processing Systems*, vol. 35, pp. 34464–34477, Curran Associates, Inc., 2022.

[176] Y. Yang, Y. Jiang, Y. Liu, J. Chen, and S. E. Li, "Model-free safe reinforcement learning through neural barrier certificate," *IEEE Robotics and Automation Letters*, vol. 8, no. 3, pp. 1295–1302, 2023.

[177] J. Dai, J. Ji, L. Yang, Q. Zheng, and G. Pan, "Augmented proximal policy optimization for safe reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 6, pp. 7288–7295, 2023.

[178] D. Schwung, F. Csaplar, A. Schwung, and S. X. Ding, "An application of reinforcement learning algorithms to industrial multi-robot stations for cooperative handling operation," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pp. 194–199, IEEE, 2017.

[179] R. Meyes, H. Tercan, S. Roggendorf, T. Thiele, C. Büscher, M. Obdenbusch, C. Brecher, S. Jeschke, and T. Meisen, "Motion planning for industrial robots using reinforcement learning," *Procedia CIRP*, vol. 63, pp. 107–112, 2017.

[180] A. Iriondo, E. Lazkano, L. Susperregi, J. Urain, A. Fernandez, and J. Molina, "Pick and place operations in logistics using a mobile manipulator controlled with deep reinforcement learning," *Applied Sciences*, vol. 9, no. 2, p. 348, 2019.

[181] A. Jafari-Tabrizi and D. P. Gruber, "Reinforcement-learning-based control of an industrial robotic arm for following a randomly-generated 2d-trajectory," in *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, pp. 1–6, IEEE, 2021.

[182] P. Egli and M. Hutter, "Towards rl-based hydraulic excavator automation," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2692–2697, IEEE, 2020.

[183] P. Egli and M. Hutter, "A general approach for the automation of hydraulic excavator arms using reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5679–5686, 2022.

[184] Q. Liu, Z. Liu, B. Xiong, W. Xu, and Y. Liu, "Deep reinforcement learning-based safe interaction for industrial human-robot collaboration using intrinsic reward function," *Advanced Engineering Informatics*, vol. 49, p. 101360, 2021.

[185] OpenAI, *Benchmarking Safe Exploration in Deep Reinforcement Learning*, 2019.

[186] A. Stooke, J. Achiam, and P. Abbeel, "Responsive safety in reinforcement learning by pid lagrangian methods," in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, pp. 9133–9143, PMLR, 2020.

[187] C. Liu and M. Tomizuka, "Control in a safe set: Addressing safety in human-robot interactions," in *Dynamic Systems and Control Conference*, vol. 46209, p. V003T42A003, American Society of Mechanical Engineers, 2014.

[188] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[189] J. Ji, B. Zhang, J. Zhou, X. Pan, W. Huang, R. Sun, Y. Geng, Y. Zhong, J. Dai, and Y. Yang, "Safety-gymnasium: A unified safe reinforcement learning benchmark," *arXiv*, vol. arXiv:2310.12567, 2023. Accepted for presentation at the 37th Conference on Neural Information Processing Systems Datasets and Benchmarks Track.

[190] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19, AAAI Press, 2019.

[191] Y. Emam, P. Glotfelter, Z. Kira, and M. Egerstedt, "Safe model-based reinforcement learning using robust control barrier functions," *arXiv preprint arXiv:2110.05415*, 2021.

[192] Y. Luo and T. Ma, "Learning barrier certificates: Towards safe reinforcement learning with zero training-time violations," *Advances in Neural Information Processing Systems*, vol. 34, pp. 25621–25632, 2021.

[193] P. Liu, D. Tateo, H. B. Ammar, and J. Peters, "Robot reinforcement learning on the constraint manifold," in *Proceedings of the 5th Conference on Robot Learning* (A. Faust, D. Hsu, and G. Neumann, eds.), vol. 164 of *Proceedings of Machine Learning Research*, pp. 1357–1366, PMLR, 08–11 Nov 2022.

[194] Z. Liu, H. Zhou, B. Chen, S. Zhong, M. Hebert, and D. Zhao, "Constrained model-based reinforcement learning with robust cross-entropy method," *arXiv preprint arXiv:2010.07968*, 2020.

La borsa di dottorato è stata cofinanziata con risorse del
Programma Operativo Nazionale Ricerca e Innovazione 2014-2020, risorse FSE REACT-EU
Azione IV.4 “Dottorati e contratti di ricerca su tematiche dell’innovazione”
e Azione IV.5 “Dottorati su tematiche Green”