

WirelessDT: A Digital Twin Platform for AI-Enabled Wireless Systems

ZHONGZHENG LAI

Doctor of Philosophy



THE UNIVERSITY OF
SYDNEY

Supervisor: Dong Yuan
Associate Supervisors: Huaming Chen, Wei Bao

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

Faculty of Engineering
The University of Sydney
Australia

12 January 2024

Statement of Originality

This is to certify that, to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and all the assistance received in preparing this thesis and sources have been acknowledged.

Zhongzheng Lai

28 September 2023

Authorship Attribution Statement

Chapter 3 of this thesis contains material from my published paper: *Lai, Zhongzheng, Dong Yuan, Wei Bao, Yu Zhang, and Bing Bing Zhou. "DeepWiSim: A Wireless Signal Simulator for Automatic Deep Learning." In Proceedings of the SIGCOMM'22 Poster and Demo Sessions, pp. 13-15. 2022.*

I designed and developed the simulator, conducted experiments, analyzed the data, and wrote the article.

Chapter 4 of this thesis contains material from my published paper: *Lai, Zhongzheng, Dong Yuan, Huaming Chen, Yu Zhang, and Wei Bao. "WirelessDT: A Digital Twin Platform for Real-Time Evaluation of Wireless Software Applications." In 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 146-150. IEEE, 2023.*

I designed and developed the platform, conducted experiments, analyzed the data, and wrote the article.

Chapter 5 of this thesis contains material from my submitted paper: *Lai, Zhongzheng, Huaming Chen, John Grundy, Yu Zhang, Dong Yuan. "WirelessDT: A Digital Twin System for Wireless and Mobile Applications." IEEE Transactions of Mobile Computing, 2023.*

I designed and developed the system, conducted experiments, analyzed the data, and wrote the article.

In addition to the statements above, in cases where I am not the corresponding author of a published item, permission to include the published material has been granted by the corresponding author.

Zhongzheng Lai

28 September 2023

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Dong Yuan

28 September 2023

Abstract

Wireless technologies play a critical role in various applications, including communications, pedestrian localization, autonomous vehicles, [Virtual Reality \(VR\)](#), [Augmented Reality \(AR\)](#), and health care. The inherent reliance of wireless systems on spatially propagating radio signals for information transmission makes them vulnerable to signal fluctuations caused by wireless sensing systems and environmental dynamics (e.g., fluctuating crowd densities and stochastic dynamic events). This characteristic makes it difficult to accurately and rationally analyze wireless signals and applications related to wireless signals. Furthermore, integrating [Artificial Intelligence \(AI\)](#) techniques into wireless systems increases the data requirements, especially for training and validating [Deep Learning \(DL\)](#) models. To ensure robust generalization of these [AI](#) models, the quality and quantity of the training data must meet rigorous standards. However, obtaining standardized, high-quality data to train and validate models in real-world settings is often labor-intensive and time-consuming. Although existing wireless signal simulation tools (e.g., Matlab, Wireless InSite) have rich signal simulation capabilities, these tools focus on theoretical research and make many simplifications to the environment models (e.g., buildings, vehicles, people), which makes it difficult to accurately respond to the impact of the dynamic environment on the wireless system and to meet the demand for high-fidelity real-time simulation and model training.

To address the need for high quality wireless training data and a highly controllable wireless analysis environment, this study introduced a groundbreaking digital twin platform, WirelessDT, designed to simulate and generate wireless signals in dynamic 3D environments in real time. Based on the digital twin architecture, WirelessDT combines a high-performance 3D game engine, a [Graphics Processing Unit \(GPU\)](#)-based [Real-Time Ray Tracing \(RTRT\)](#) rendering pipeline, and adaptive learning algorithms to achieve breakthroughs in data fidelity, application efficiency, and system scalability.

WirelessDT revolutionizes wireless signal propagation simulation by leveraging the powerful capabilities of [Unreal Engine 5 \(UE5\)](#), a leading 3D engine originally developed for game design. By pioneering the innovative use of the [GPU RTRT](#) rendering pipeline for wireless propagation analysis, the real-time wireless simulation in WirelessDT is able to match the performance of real-time graphical rendering. This breakthrough in real-time performance allows WirelessDT to seamlessly integrate with the digital twin technology, thus ensuring a seamless convergence of the real and virtual environments.

In addition, this research introduces a novel learning-based calibration technique that enhances the realism and accuracy of simulated data by using real-world data to fine-tune model parameters. This breakthrough algorithm views the entire digital twin system as a generative model, and in conjunction with the parametric design of WirelessDT, the algorithm can adjust the system parameters based on the training data and effectively improve the simulation fidelity of the data.

These cutting-edge features greatly expand the utility of WirelessDT, making it an indispensable tool for a wide variety of applications that require interactive simulation and immediate feedback. In addition, a thorough evaluation has been conducted, demonstrating the remarkable performance of WirelessDT in terms of data accuracy, application efficiency, and system scalability.

Acknowledgements

Looking back on the four years of my doctoral journey, it was anything but smooth sailing. Throughout this transformative period, I found myself constantly wavering between self-affirmation and self-doubt. From the outset, where I embarked on this journey with unbridled confidence, to the challenging mid-phase marred by repeated setbacks and introspection, and finally to the triumphant end where I overcame numerous obstacles, the unwavering support of my mentors, peers, and family was instrumental.

First, I extend my deepest gratitude to my supervisor, Dr. Dong Yuan. His invaluable guidance, steadfast support, and mentorship have been pivotal in shaping my research. His profound expertise and insights were indispensable in bringing my thesis to fruition. I also wish to express my heartfelt appreciation to Dr. Huaming Chen. His invaluable contributions, informative discussions, and depth of expertise have significantly enriched my research journey. His critiques and perspectives provided much-needed clarity and direction. I am equally grateful to my associate supervisor, Dr. Wei Bao, for his invaluable advice, constructive feedback, and encouragement.

To my parents, I am in immeasurable debt of gratitude. Their boundless love, sacrifices, and unwavering faith in my potential have been the guiding stars of my academic journey. Their prayers, encouragement, and endless support have been the bedrock upon which I built my achievements.

Last, but by no means least, my profound thanks go to my wife. Her enduring patience, understanding, and love have been my sanctuary. Her unwavering belief in my capabilities and her relentless encouragement, especially during the most challenging times, have been the pillars of strength that propelled me forward.

This invaluable experience in my life will undoubtedly guide and inspire me as I continue on my future journey.

List of Publications

Publications or accepted paper:

- [1] Zhongzheng Lai, Dong Yuan, Huaming Chen, Yu Zhang, and Wei Bao. 2023. Wirelesdt: A digital twin platform for real-time evaluation of wireless software applications. *In 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 146–150. (Chapter 4)
- [2] Zhongzheng Lai, Dong Yuan, Wei Bao, Yu Zhang, and Bing Bing Zhou. 2022. Deepwisim: a wireless signal simulator for automatic deep learning. *In Proceedings of the SIGCOMM’22 Poster and Demo Sessions*, pages 13–15. (Chapter 3)
- [3] Handling Heavy Occlusion in Dense Crowd Tracking by Focusing on the Heads. Yu Zhang, Huaming Chen, Zhongzheng Lai, Zao Zhang and Dong Yuan. 2023. *AI 2023: Advances in Artificial Intelligence*.
- [4] Zhang, Yu, Zhongzheng Lai, Dong Yuan, Wei Bao, Bing Bing Zhou, Jing Qiu, Shen Wang, and Stewart Adams. 2021. GauPro: An Accuracy-Improved Indoor Positioning System Based on Beacon Probabilistic Fingerprint. *In 2021 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 276-283.

Submitted paper:

- [1] Zhongzheng Lai, Huaming Chen, John Grundy, Yu Zhang, and Dong Yuan. 2024. WirelesDT: A Digital Twin System for Wireless and Mobile Applications. *IEEE Transactions on Mobile Computing*.(Chapter 5)
- [2] Zhongzheng Lai, Huaming Chen, Ruoxi Sun, Yu Zhang, Minhui Xue and Dong Yuan. 2024. On Security Weaknesses and Vulnerabilities in AI-Enabled Systems. *IEEE Transactions on Dependable and Secure Computing*.

Contents

Statement of Originality	i
Authorship Attribution Statement	ii
Abstract	iv
Acknowledgements	vi
List of Publications	vii
Glossary	xiii
List of Figures	xvii
List of Tables	xix
Chapter 1 Introduction	1
1.1 Motivation	2
1.2 Background	3
1.3 Proposed Approaches	5
1.4 Contribution	6
1.5 Thesis Outline	7
Chapter 2 Literature Review	10
2.1 Digital Twin	10
2.1.1 Concept of Digital Twin	11
2.1.2 Enabling Technologies for Digital Twin	14
2.1.3 Digital Twin Applications	18
2.1.4 Potentials and Challenges	20
2.2 AI-Enabled Wireless Systems	22

2.2.1	Evolution: From Maxwell's Equation to Smart Wireless Systems	23
2.2.2	AI-Enabled Wireless Systems for Indoor Positioning	26
2.3	Wireless Signal Simulation	40
2.3.1	The Role of Simulation	41
2.3.2	Wireless Signal Propagation	44
2.3.3	Wireless Propagation Modeling	47
2.4	Ray Tracing	49
2.4.1	The Concept of Ray Tracing	49
2.4.2	Ray Tracing Algorithms	50
2.4.3	Ray Tracing Applications	53
Chapter 3	Digital Twin Platform for AI-Enabled Wireless Systems	55
3.1	Introduction	55
3.2	Related Work	57
3.3	The Design of WirelessDT	58
3.3.1	Physical Twin	60
3.3.2	Digital Twin	63
3.3.3	Communication Interface	66
3.4	Proof-of-Concept Implementation of WirelessDT	67
3.4.1	Implementing the Physical Twin	67
3.4.2	Implementing the Digital Twin	69
3.5	Summary	71
Chapter 4	GPU Real-Time WiTracing	73
4.1	Introduction	73
4.2	Related Work	75
4.2.1	Real-Time Engine-Based Simulation	75
4.2.2	Ray Tracing-Based Wireless System Simulation	75
4.3	Preliminary	76
4.3.1	Ray Tracing in Rendering Engine	76
4.3.2	Ray Tracing Pipeline	77

4.3.3	Rendering by Ray Tracing	78
4.4	Real-Time WiTracing Engine	79
4.4.1	Sensor Simulation	79
4.4.2	Wireless Propagation Simulation	80
4.4.3	WiTracing Algorithm	83
4.4.4	Parallel Data Aggregation	85
4.4.5	Data Caching and Visualization	86
4.5	Learning-Based Calibration	87
4.5.1	Calibration with Real World Data	88
4.5.2	Formulation	90
4.5.3	Training	91
4.6	Proof-of-Concept Implementation of WiTracing	96
4.6.1	Extending Render Module for WiTracing	96
4.6.2	Perform WiTracing in Shaders	105
4.7	Summary	116
Chapter 5	Experimental Evaluation of WirelessDT	117
5.1	Introduction	117
5.2	Fidelity Evaluation	119
5.2.1	Experimental Dataset	119
5.2.2	Environmental Modeling	121
5.2.3	Feature Engineering	122
5.2.4	Training WirelessDT	132
5.2.5	Experimental Results	138
5.3	Practicality Evaluation	140
5.3.1	Environmental Modeling	141
5.3.2	Example Application	143
5.3.3	Results and Discussion	144
5.4	Scalability Evaluation	146
5.5	Summary	148

Chapter 6 Conclusion	150
6.1 Limitation Discussion	150
6.2 Conclusion	151
References	153

Glossary

5G: Fifth Generation. [25](#), [75](#)

6G: Sixth Generation. [25](#)

AHRS: Attitude and Heading Reference System. [68](#), [120](#)

AI: Artificial Intelligence. [iv](#), [11](#), [20](#), [22](#), [25](#), [64](#), [66](#), [67](#), [70](#), [142](#), [143](#)

AoA: Angle of Arrival. [30–32](#), [36](#), [61](#)

AP: Access Point. [127](#), [129](#), [130](#), [132–135](#)

API: Application Programming Interface. [66](#), [67](#), [96](#)

AR: Augmented Reality. [iv](#), [11](#), [17](#)

BLE: Bluetooth Low Energy. [32](#), [35](#), [61](#), [68](#), [89](#)

BRDF: Bidirectional Reflectance Distribution Function. [52](#), [82](#), [84](#), [112](#)

BVH: Bounding Volume Hierarchy. [78](#)

CDF: Cumulative Distribution Function. [112](#)

CFR: Channel Frequency Response. [29](#)

CIR: Channel Impulse Response. [29](#), [31](#)

CPU: Central Processing Unit. [4](#), [75](#), [102](#), [146](#), [147](#)

CSI: Channel State Information. [28](#), [29](#), [32](#), [61](#)

dBm: decibel milliwatts. [28](#)

DL: Deep Learning. [iv](#), [2](#), [15](#), [17](#), [22](#), [25](#), [27](#), [40](#), [96](#)

DNN: Deep Neural Network. [40](#)

DRNN: Deep Recurrent Neural Network. [143](#), [144](#)

ED: Euclidean Distance. [91](#), [92](#), [132](#), [134](#), [135](#)

EM: Electromagnetic. [23–25](#)

FDTD: Finite-Difference Time-Domain. [47, 48](#)

FM: Frequency Modulation. [27](#)

FPS: Frames Per Second. [147, 148](#)

GNSS: Global Navigation Satellite System. [120](#)

GPS: Global Positioning System. [14](#)

GPU: Graphics Processing Unit. [iv, v, 1, 3, 4, 6, 9, 56, 64, 65, 73–77, 79, 80, 83, 96, 99, 101, 102, 118, 151](#)

GTD: Geometrical Theory of Diffraction. [50](#)

GUI: Graphical User Interface. [144](#)

HIL: Hardware-in-the-Loop. [7, 66, 75](#)

HMI: Human-Machine Interface. [14, 17](#)

IMU: Inertial Measurement Unit. [14, 34, 35](#)

IoDT: Internet of Digital Twin. [11, 17](#)

IoT: Internet of Things. [11, 14, 16, 20, 22, 25](#)

IPIN: Indoor Positioning and Indoor Navigation. [119, 121, 123, 126, 127](#)

IPS: Indoor Positioning System. [27, 28, 35, 37, 38](#)

JS Divergence: Jensen-Shannon Divergence. [91, 92, 130–132, 134, 135, 138–140](#)

KF: Kalman Filter. [140, 143, 144](#)

KL Divergence: Kullback-Leibler Divergence. [89, 92, 138, 139](#)

KNN: K-Nearest Neighbor. [40](#)

LED: Light Emitting Diode. [34](#)

LiDAR: Light Detection and Ranging. [14, 75, 146](#)

LoS: Line-of-Sight. [28, 30, 34, 36, 50](#)

MAC address: Media Access Control Address. [126, 127, 129–131](#)

MAE: Mean Absolute Error. [139](#)

MAP: Maximum A Posteriori. [39](#)

MIMO: Multiple-Input Multiple-Output. [31](#)

MIS: Multiple Importance Sampling. [82](#), [84](#)

ML: Machine Learning. [14–16](#), [22](#), [27](#), [39](#), [40](#), [58](#), [70](#)

MMSE: Minimum Mean Square Error. [39](#)

MSE: Mean Square Error. [89](#), [139](#)

NFC: Near Field Communication. [68](#)

NIC: Network Interface Card. [130](#)

NLoS: Non-Line-of-Sight. [30](#)

NPC: Non-Player Character. [4](#), [64](#), [70](#)

PDF: Probability Density Function. [38](#)

PLM: Product Lifecycle Management. [11](#), [12](#), [14](#)

PoC: Proof-of-Concept. [7](#)

RADAR: Radio Detecting And Ranging. [14](#), [75](#), [146](#)

RDG: Render Dependency Graph. [101](#), [102](#)

RFID: Radio Frequency Identification. [27](#), [31](#), [33](#)

RL: Reinforcement Learning. [15](#)

RNN: Recurrent Neural Network. [140](#)

RSS: Received Signal Strength. [28](#), [29](#), [32](#), [35](#), [37–39](#), [61](#), [88](#), [89](#), [127](#), [130](#), [132–134](#), [138](#), [143](#), [144](#)

RSSI: Received Signal Strength Indicator. [28](#)

RT: Ray Tracing. [4–7](#), [9](#), [10](#), [47–54](#), [56](#), [65](#), [74–82](#), [96](#), [100](#), [101](#), [106](#), [111](#), [116–118](#), [146–148](#), [151](#)

RTRT: Real-Time Ray Tracing. [iv](#), [v](#), [1](#), [3](#), [5](#), [6](#), [9](#), [56](#), [58](#), [64](#), [65](#), [73–77](#), [79](#), [80](#), [96](#), [100](#), [116](#), [118](#), [150](#), [151](#)

RTT: Round Trip Time of Arrival. [29–31](#), [35](#)

Rx: Receiver. [81–83](#), [90](#), [96](#), [101](#), [102](#), [106](#), [112](#), [141–143](#)

SBR: Shooting and Bouncing Ray. [51](#), [52](#)

SIL: Software-in-the-Loop. [7](#), [67](#), [75](#), [76](#), [87](#), [145](#)

TDoA: Time Difference of Arrival. [29–31](#), [35](#), [38](#)

ToA: Time of Arrival. [29–32](#), [34](#), [35](#), [38](#), [61](#)

Tx: Transmitter. [81](#), [82](#), [84](#), [96](#), [101](#), [102](#), [106](#), [141](#), [142](#)

UDP: User Datagram Protocol. [70](#)

UE5: Unreal Engine 5. [v](#), [1](#), [4](#), [6](#), [54](#), [56](#), [64](#), [65](#), [70](#), [75–81](#), [96](#), [105](#)

UJI Library: Biblioteca Universitat Jaume I. [119](#), [121](#), [122](#)

USF: Unreal Shader File. [96](#)

USH: Unreal Shader Header. [96](#)

UTD: Uniform Theory of Diffraction. [50](#), [54](#)

UWB: Ultra-wideband. [27–29](#), [31](#), [61](#), [72](#)

VLC: Visible Light Communication. [34](#)

VR: Virtual Reality. [iv](#), [11](#), [17](#)

WLAN: Wireless Local Area Network. [31](#), [68](#)

ZigBee: Zonal Intercommunication Global-standard. [27](#), [32](#), [68](#)

List of Figures

1.1	Thesis outline.	8
2.1	An overview digital twins to empower smart city applications.	11
2.2	Conceptual framework of the digital twin originating from the 2002 PLM center proposal. Adapted from (Grieves and Vickers, 2017).	12
2.3	AI-enabled wireless network system.	26
2.4	An AI-enabled wireless positioning system.	26
2.5	Plane wave polarizations. Adapted from (Zaidi <i>et al.</i> , 2018).	45
3.1	Architecture of WirelessDT.	59
3.2	Architecture of physical twin module in WirelessDT.	60
3.3	Architecture of digital twin module in WirelessDT.	63
3.4	Physical twin service.	68
3.5	The proof-of-concept implementation of physical twin.	69
3.6	Digital twin components.	69
4.1	DirectX ray tracing pipeline. The potential recursive calls are not shown. Adapted from (Haines and Akenine-Möller, 2019).	77
4.2	The octahedral mapping.	80
4.3	The process of WiTracing.	81
4.4	The multi-layer downsampling for data aggregation.	86
4.5	The visualization of WiTracing results.	86
5.1	Biblioteca Universitat Jaume I (UJI Library).	119
5.2	Indoor maps of UJI library from level 1 to 5.	120
5.3	Data collection process for IPIN dataset.	122
5.4	Building model of UJI library from level 2 to 5.	123
5.5	Comparison of virtual scene and real scene.	124
5.6	WiFi measurement in IPIN dataset.	124

5.7	Interpolation of WiFi measurement position for single trajectory.	125
5.8	Interpolation of WiFi measurement position for MAC 20:20:00:00:01:50 at level 2.	126
5.9	RSS distribution for MAC 20:20:00:00:01:50 at different floor.	128
5.10	Search for optimal n value for floor detection.	128
5.11	The number of measurement for each MAC (with quantity greater than 300).	129
5.12	RSS probability distribution over different MAC address.	130
5.13	Cross comparison of RSS measurement distribution. White area indicate the MAC addresses is not belong to access point from same floor.	131
5.14	Search similar distribution with JS divergence.	131
5.15	The overview of training process.	133
5.16	Determining access point coordinates boundary conditions.	134
5.17	Loss space visualization cross penetration factor, reflection factor and transmit power.	136
5.18	Loss space visualization cross signal source position.	137
5.19	Comparison of received signal strength distribution on different level.	141
5.20	Online calibration in small scale corridor environment.	142
5.21	Testing environment of large-scale office scene.	143
5.22	The evaluation of wireless positioning application.	145
5.23	Variation of frame rate with the number of transceiver pairs performing WiTracing in a single frame under different sensor resolution conditions.	147

List of Tables

2.1 The Definition of Digital Twin from Literature	13
2.2 Indoor Localization Technologies	31
3.1 Physical twin hardware components.	68
5.1 The experimental result of fidelity evaluation for WirelessDT(WDT) and Baseline (BL)	140
5.2 The experimental result of scalability evaluation	146

CHAPTER 1

Introduction

This thesis introduces a groundbreaking digital twin platform, WirelessDT, designed for real-time simulation and data generation of wireless signals within dynamic 3D environments. WirelessDT is built on the widely used [UE5](#), a leading game engine. The platform harnesses the potent potential of game engine and its highly optimized [GPU RTRT](#) pipeline to create 3D dynamic scenes and simulate wireless signal propagation. Initially, the [GPU RTRT](#) pipeline was intended for rendering real-time scenes. However, by innovatively adapting the [GPU RTRT](#) rendering pipeline to facilitate wireless propagation simulation, WirelessDT can deliver competitive performance comparable to real-time graphical rendering. This superior real-time performance enables WirelessDT to seamlessly align with digital twin architecture, facilitating synchronization between reality and the virtual world. In addition, this thesis introduces a novel learning-based calibration technique that enhances the realism and accuracy of simulated data by using real-world data to fine-tune model parameters. This breakthrough algorithm views the entire digital twin system as a generative model. In conjunction with the parametric design of WirelessDT, the algorithm can adjust the system parameters based on the training data and effectively improve the simulation fidelity of the data. These advanced capabilities broaden the applicability of WirelessDT, making it highly useful for a wide range of applications that require interactive simulations and real-time evaluation. Thus, the introduction of WirelessDT marks a significant step forward in wireless signal simulation, promising improved utility and efficiency in diverse application contexts.

1.1 Motivation

Wireless technology is crucial in various fields, such as pedestrian localization, self-driving cars, unmanned aerial vehicles, extended reality, and healthcare. The integration of DL has expanded these systems' capabilities, with many incorporating DL models to enhance performance (Zhang *et al.*, 2019).

The merger of AI and wireless systems with DL models creates a high demand for extensive data essential for training effective DL models. Acquiring large volumes of high-quality data is challenging due to unpredictable factors in real-world scenarios, such as fluctuating crowd dynamics and unexpected events.

In sectors like autonomous driving and unmanned aerial vehicles, advanced simulators (Shah *et al.*, 2018; dSpace, 2023; Dosovitskiy *et al.*, 2017) have been extensively utilized for model training, validation, and algorithmic development, often with notable success. However, limited by their specific design objectives, these simulators do not address the broader needs of AI-enabled wireless systems. Consider the simulation of an indoor wireless positioning scenario: it requires a large-scale, high-performance simulation framework. A quintessential wireless system, such as an indoor positioning system, is typified by a wide array of wireless transmitters and receivers that could reach hundreds or thousands in real-world deployments. For a simulation to be of tangible benefit, it must adeptly mimic signal propagation across this extensive network of transceivers. Furthermore, considering the environmental sensitivity of wireless sensor networks and their vulnerability to significant signal variations due to dynamic factors such as human movement (He and Chan, 2015; Zafari *et al.*, 2019), a simulation tool must be equipped to factor in these dynamics. Unfortunately, current simulation tools are deficient in these areas.

Although traditional wireless simulation tools like MATLAB (MathWorks, 2023), CST Studio Suite (Simulia, 2020), Wireless InSite (Remcom, 2023) are comprehensive for signal simulation, they are more suited for theoretical research and tend to oversimplify environmental models. This results in difficulties in accurately simulating dynamic environmental impacts

on wireless systems, particularly for high-fidelity, real-time simulations, and model training. Moreover, their efficiency needs to be improved for real-time applications.

This leads to a key research problem: How can we efficiently produce high-fidelity simulation data in real-time for training, testing, and evaluating AI-enabled wireless systems?

To address these issues, this these introduced a groundbreaking digital twin platform, WirelessDT, designed to simulate and generate wireless signals in dynamic 3D environments in real time. Based on the digital twin architecture, WirelessDT combines a high-performance 3D game engine, a GPU-based RTRT rendering pipeline, and adaptive learning algorithms to achieve breakthroughs in data fidelity, application efficiency, and system scalability, which allow it the satisfy the required performance for AI-enabled wireless system training and wireless analytical tasks.

1.2 Background

Over the years, traditional wireless simulation tools such as MATLAB (MathWorks, 2023), CST Studio Suite (Simulia, 2020), Wireless InSite (Remcom, 2023), WinProp (ALTAIR, 2023), Network Simulator 3 (NS-3) (Riley and Henderson, 2010), and OMNeT++ (Varga, 2010) have been instrumental in advancing wireless signal system research. This includes areas such as wireless signal propagation analysis, wireless communication, routing protocols, and Internet systems. MATLAB (MathWorks, 2023), a versatile tool for numerical computation, visualization, and programming, offers a comprehensive environment for end-to-end wireless communication system simulations when paired with Simulink and the Communications System Toolbox. On the other hand, CST Studio Suite (Simulia, 2020), Wireless InSite (Remcom, 2023), and WinProp (ALTAIR, 2023) are adapted for simulations of electromagnetic and wireless signals. While CST Studio Suite (Simulia, 2020) and Wireless InSite (Remcom, 2023) delve deeper into physical aspects such as antenna design, WinProp (ALTAIR, 2023) emphasizes wave propagation across diverse environments. NS-3 (Riley and Henderson, 2010) and OMNeT++ (Varga, 2010), both open-source network simulators, differ in their focus: NS-3 caters to a broad spectrum of network protocols, whereas OMNeT++

adopts a modular approach, making it adaptable to a variety of applications beyond mere networking.

However, these tools need to meet the demands of large-scale, real-time, dynamic wireless simulations essential for AI-driven wireless systems. While MATLAB (MathWorks, 2023) boasts robust wireless signal analysis libraries, it lacks support for real-time simulations in dynamic contexts due to its reliance on Central Processing Unit (CPU)-based Ray Tracing (RT). CST Studio Suite (Simulia, 2020), Wireless InSite (Remcom, 2023), and WinProp (ALTAIR, 2023), despite their commendable simulation and GPU-based computing capability, do not meet the real-time simulation criterion. Furthermore, their deployment demands intricate pre-configuration and specialized expertise. NS-3 (Riley and Henderson, 2010) and OMNeT++ (Varga, 2010), with their pronounced emphasis on internal signal network simulations, overlook the crucial aspect of external environmental impacts on wireless signal systems. This underscores the pressing need for a novel wireless signal simulation system tailored to AI-integrated wireless systems' data demands.

Currently, the evolution of 3D game engines, such as UE5 (EpicGames, 2023) and Unity (UnityTechnologies, 2023), has revolutionized real-time 3D dynamic scene simulations. These engines have been harnessed in numerous studies to craft and operationalize digital twin systems (Harrington *et al.*, 2022; Tao *et al.*, 2022). Inherently designed for dynamic simulations, game engines excel in realistic environment modeling, versatile human character design, intelligent Non-Player Characters (NPCs) control, and rudimentary simulations of physical phenomena. Their efficient and optimized architecture is conducive to large-scale simulation scenarios. Furthermore, RT, a deterministic method, has gained traction in wireless signal propagation simulations and channel modeling (He *et al.*, 2019). Its merits, including minimal bandwidth and carrier frequency dependence, superior spatial resolution, ability to transcend channel measurement and stochastic method limitations, and compatibility with other modeling techniques, have positioned ray tracing as a preferred choice for wireless signal simulator development.

By synergizing these established and emergent technologies (digital twins, 3D real-time rendering, ray-tracing algorithms), this study have conceptualized and realized WirelessDT, a simulation system adept at large-scale, high-fidelity, real-time wireless signal simulations.

1.3 Proposed Approaches

The primary goal of this study is to advance AI-enhanced wireless system technology through innovative data collection and simulation methods. This research aims to surpass traditional wireless emulators' constraints in executing large-scale, high-fidelity, real-time simulations. This study focus on integrating digital twin technology, 3D rendering engines, and real-time ray tracing (RTRT) techniques. This integration fosters a more efficient and realistic simulation environment, effectively narrowing the discrepancy between virtual simulations and actual wireless system performance. In pursuit of these goals, this thesis introduces specific methodologies and tools to transform wireless system simulation and evaluation:

- **WirelessDT Development:** This research introduces WirelessDT, a cutting-edge digital twin platform for wireless systems. WirelessDT is tailored to offer high fidelity, efficiency, and scalability. It allows real-time interaction with physical wireless systems, promoting seamless integration and synchronization with existing networks.
- **Adaptive Learning in WirelessDT:** WirelessDT incorporates a dynamic learning mechanism that uses real-world data to refine its simulation parameters continuously. This feature ensures high-fidelity simulations, especially in complex environments like indoor spaces, closely aligning simulated outcomes with real-world data.
- **WiTracing Engine – Enhancing Simulation Efficiency:** This study developed the WiTracing Engine, a bespoke [RTRT](#) rendering pipeline, to address the need for more efficient simulation processes. This engine utilizes [RT](#) technology for wireless signal simulation, achieving the efficiency of light rendering in gaming engines and meeting the demands of real-time simulation.

- **Utilizing Advanced 3D Engine:** By employing a robust 3D engine, WirelessDT gains unparalleled flexibility and compatibility. This choice directly supports the objective of creating dynamic and realistic simulation environments. It allows quick adaptation to different scenarios, from static model creation to dynamic simulations of moving entities, like crowds or vehicles.

1.4 Contribution

The primary objective of this thesis is to architect and realize a high-performance digital twin platform tailored to simulate wireless signal propagation and generate pertinent data. Our vision is to craft a holistic framework that facilitates the evaluation of AI-enhanced wireless systems across their entire lifecycle.

In conceptualizing and realizing WirelessDT, this study seamlessly merged the digital twin architecture, [RT](#) technologies, and the game engine's capabilities. The choice of [UE5](#) as the foundational platform was driven by two pivotal considerations. First, [UE5](#) extends open source access to its user base, allowing us the flexibility to tailor and refine the core source code. Second, the engine boasts an integrated [GPU RTRT](#) rendering mechanism. By tweaking the inherent [RT](#) algorithm, this study utilized the [GPU RTRT](#) rendering pipeline within the game engine to emulate wireless signal propagation. This strategic use of the game engine enabled WirelessDT to fully leverage the optimized [RT](#) architecture for the simulation of wireless signals, ensuring that WirelessDT can meet performance benchmarks. To realize the overarching goal, this study meticulously endowed WirelessDT with the following salient features.

- **Wireless system simulation:** This aspect encompasses signal propagation simulation (Chapter [4](#)) and wireless device simulation (Chapter [3](#)). Signal propagation simulation addresses complex signal interactions, including direct or indirect transmission, reflection, and scattering. Conversely, the wireless device model incorporates hardware attributes like transmission power and antenna patterns. Ensuring

the precision of both these models is paramount to vouch for the authenticity and reliability of the simulated results.

- **Dynamic environment simulation:** As wireless signal propagation is significantly affected by environmental dynamics and architecture, modeling these aspects accurately is essential for realistic simulations (Chapter 3).
- **Automatic calibration:** To ensure high-fidelity simulations, WirelessDT incorporates an advanced calibration mechanism that aligns simulated outcomes with real-world scenarios, minimizing the need for manual adjustments (Chapter 4).
- **Data sharing interface:** WirelessDT includes both hardware and software interfaces to facilitate system calibration and integration with external tools and real-time applications (Chapter 3).
- **Comprehensive evaluation framework:** The platform provides an extensive framework for intuitive data interaction with real wireless systems, supporting applications such as online training, instant evaluation, and both [Software-in-the-Loop \(SIL\)](#) and [Hardware-in-the-Loop \(HIL\)](#) testing (Chapter 3).
- **Open source:** To encourage further research and customization, WirelessDT is made available as an open-source resource.

1.5 Thesis Outline

Chapter 1 introduces the foundational concepts behind WirelessDT. It delves into the motivations driving this research and offers a concise overview of the research context, the challenges encountered, and the solutions proposed. The chapter ends with an overview of the pivotal contributions of this study.

Chapter 2 delves into the background, related work, and preliminary studies. It zeros in on four pivotal domains: digital twins, intelligent wireless systems, wireless signal simulation, and [RT](#) techniques.

Chapter 3, the intricate design and [Proof-of-Concept \(PoC\)](#) implementation of the WirelessDT architecture are elucidated. Chapter 3 first sheds light on the overarching design principles

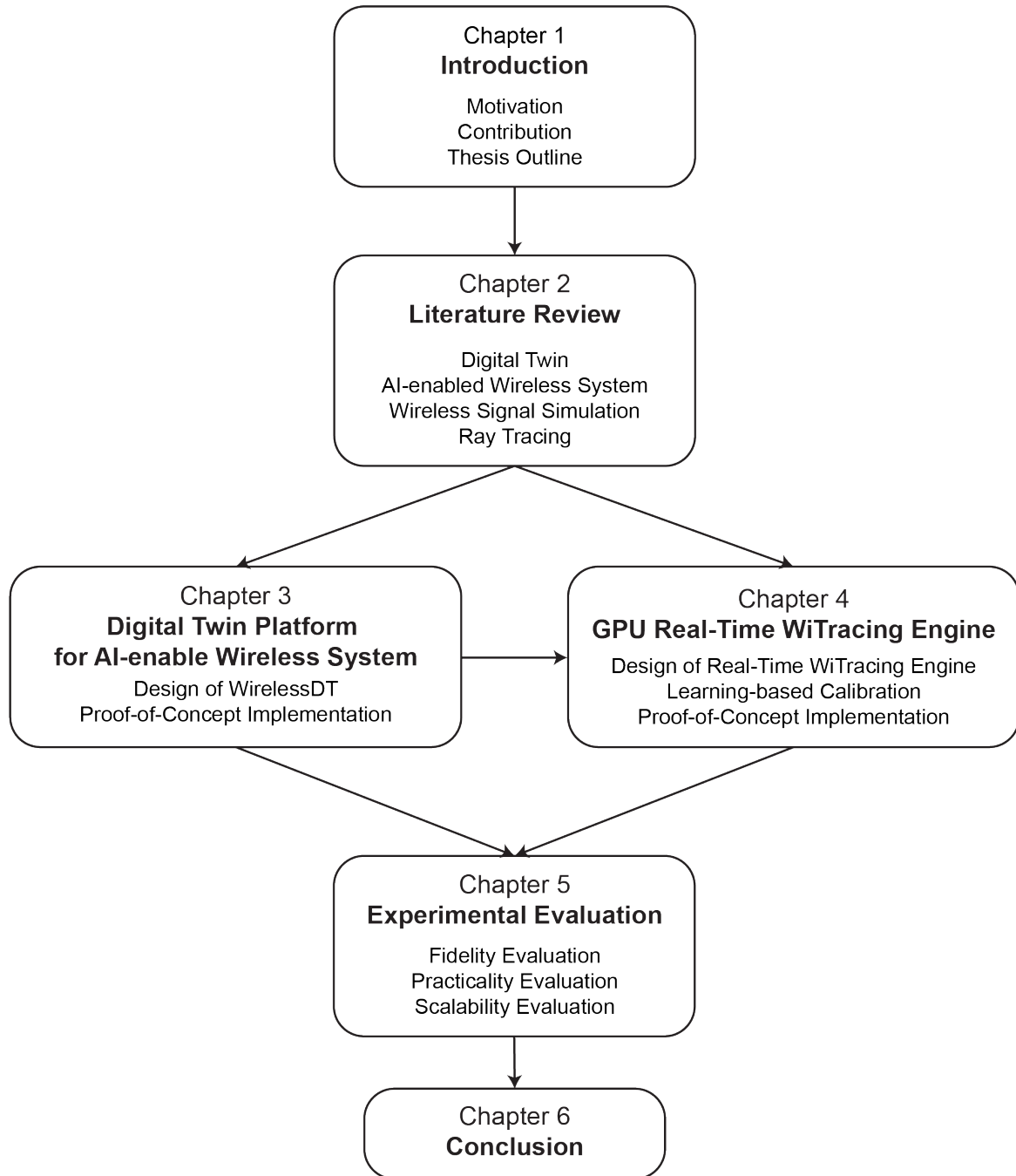


FIGURE 1.1: Thesis outline.

and their inherent advantages. Subsequently, it elaborates on the design concepts that underlie each of the three core modules: the digital twin, the physical twin, and the communication interface. The latter part of the chapter provides insight into the implementation process and methodologies tailored for these three modules.

Chapter 4 offers an in-depth exploration of real-time WiTracing, a cornerstone technology for the WirelessDT system. Beginning with the rudiments of RT, the chapter meticulously explains the fusion of wireless signal simulation with RT rendering. This amalgamation facilitates the simulation of wireless signal propagation using GPU RTRT technology. The chapter also unveils a learning-based parameter calibration technique, empowering WirelessDT to enhance simulation precision by harnessing real data. The latter segments detail the practical implementation of our proposed method within a 3D engine, intertwining algorithms, and code execution.

Chapter 5 assesses the efficacy of WirelessDT in multiple aspects, including fidelity, practicality, and scalability of the data. The chapter commences by detailing the creation of a virtual environment mirroring a tangible physical space, leveraging a genuine public dataset. It underscores the data fidelity of WirelessDT by juxtaposing simulated data from the virtual realm with actual data. Furthermore, the chapter unveils a real-world localization application scenario crafted by this research, showcasing the versatility of WirelessDT through tangible use cases like system calibration and application assessment. The chapter concludes by juxtaposing the stellar performance of WirelessDT against its contemporaries.

Chapter 6 offers a concise recapitulation of the study, spotlighting its seminal contributions and charting the trajectory for future research endeavours.

CHAPTER 2

Literature Review

This thesis is built on a solid foundation of digital twin and wireless system dynamic environment simulation: The trajectory of wireless signals is a profoundly interesting study. This chapter provides the traces the evolution of studies in the field, highlighting major milestones and seminal works that have shaped understanding. Section 2.1 presents the evaluation of the digital twin concept, technologies, applications, and challenges. Section 2.2 gives an overview of the evolution of the AI-enabled wireless system and its classic application in the indoor position dominated by the spatial-temporal dynamics and architecture. Section 2.2. Section 2.3 introduces the important theory and study for the simulation of wireless signals. Section 2.4 focuses on the fundamentals of the RT technique and its important role in the field of wireless signal simulation.

2.1 Digital Twin

Digital twin technology, an emerging and rapidly advancing field, promises substantial potential for various industrial applications, including but not limited to aerospace, construction, automotive, energy, infrastructure, government, luxury goods, manufacturing, and retail sectors (Unity, 2023; Mihai *et al.*, 2022).

The core concept of a digital twin revolves around constructing a detailed digital replica that mirrors real-world components and dynamics. In operation, the physical system and its digital counterpart exchange data through real-time communication, allowing seamless monitoring, evaluation, and forecasting of real-world systems (Singh *et al.*, 2021; Mihai *et al.*, 2022). This achievement is made possible through the strategic integration of various technologies such

as AI, 3D modeling, Internet of Things (IoT), AR, VR, and sensor technologies (Mihai *et al.*, 2022).

At a macroscopic level, Internet of Digital Twin (IoDT), represented in Figure 2.1, enables the interconnection of myriad physical entities and their corresponding digital twins. This facilitates unrestricted data exchange, dynamic collaboration on tasks, and effective information aggregation (Wang *et al.*, 2023). Such a configuration fosters a comprehensive understanding across a broad spectrum of physical and virtual entities, thereby accelerating the evolution of modern cities towards a future characterized by enhanced intelligence, predictability, and efficiency.

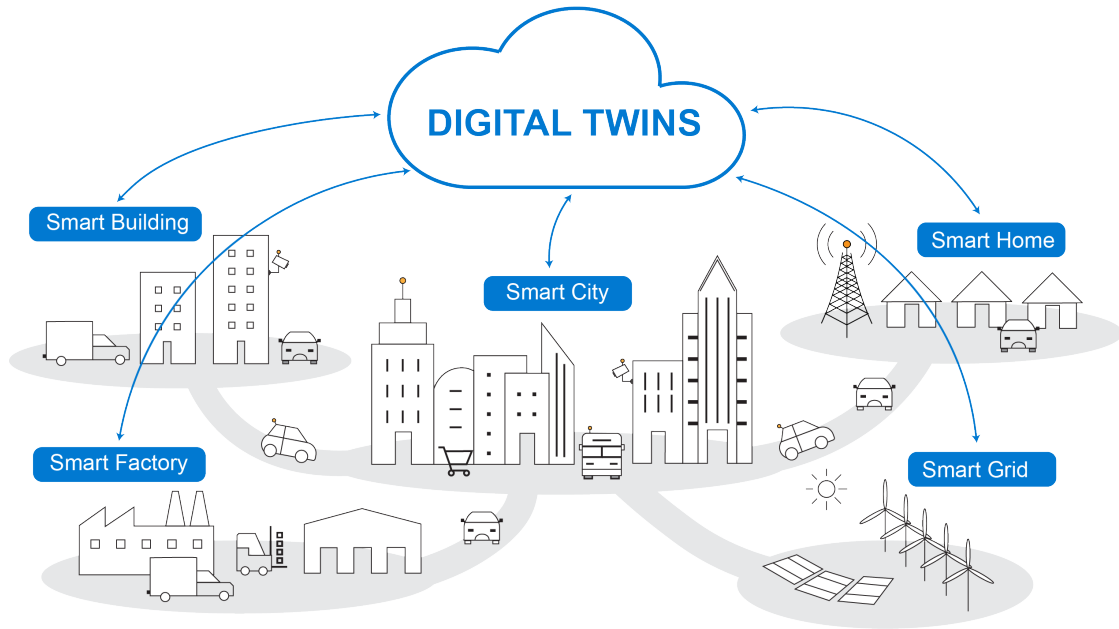


FIGURE 2.1: An overview digital twins to empower smart city applications.

2.1.1 Concept of Digital Twin

The origins of the digital twin concept can be dated back to a presentation made at the University of Michigan in 2002. Here, the framework was initially introduced as the "Conceptual Ideal for Product Lifecycle Management (PLM)" (Grieves and Vickers, 2017). This important presentation established the foundation for the core elements of the digital twins. It outlined the representation of physical space, its corresponding virtual space, and the bidirectional

communication links between these two realms. This concept acted as a precursor, paving the way for further development and broad applications of the digital twin models. Refer to Figure 2.2 for a visual representation of the original PLM conceptual model.

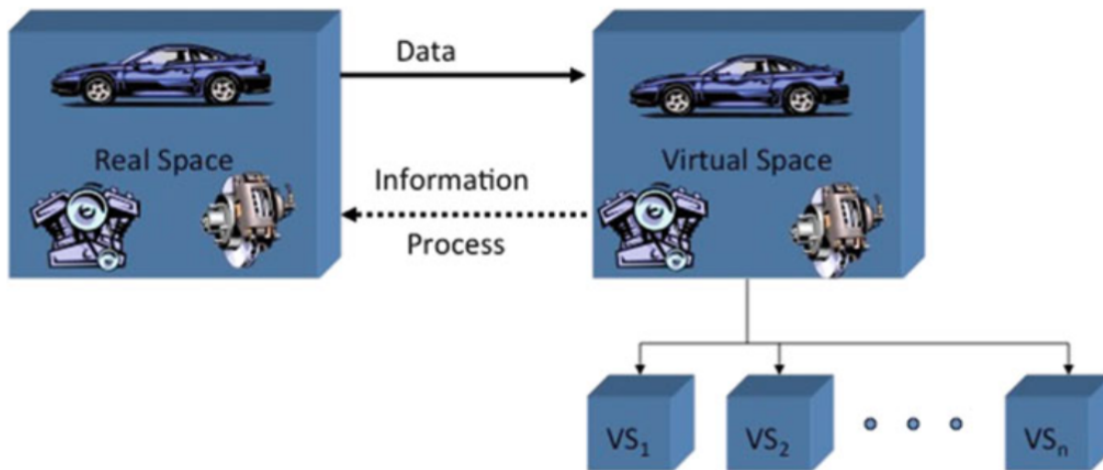


FIGURE 2.2: Conceptual framework of the digital twin originating from the 2002 PLM center proposal. Adapted from (Grieves and Vickers, 2017).

The central premise of the “Conceptual Ideal for PLM” entails the dual existence of a physical system and a corresponding virtual system, with the latter encapsulating comprehensive information of the physical system. This framework fosters a strict mirroring relationship between the virtual and real-world systems, thus bridging the gap between the physical and virtual domains. Moreover, the model underscores the dynamic interlinkages between virtual and real systems throughout all stages of their lifecycle, spanning from creation and production to operation and disposal.

Later on, in 2005, the terms “Mirrored Spaces Model” (Grieves, 2005b) and “Information Mirroring Model” (Grieves, 2005a) were introduced to represent the concept of the “Digital twin”. The term digital twin appeared for the first time on NASA’s technology roadmaps (Shafto *et al.*, 2012) and was officially formalized in a white paper by Grieves (2014), marking a key milestone in the evolution of this paradigm. Since the concept of “Digital Twin” was proposed, it has been continuously redefined with the development of science and technology and the expansion of its application fields, as shown in Table 2.1.

TABLE 2.1: The Definition of Digital Twin from Literature

Reference	Definition
(Shafto <i>et al.</i> , 2012)	"...is an integrated multi-physics, multi-scale, probabilistic simulation of a vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its flying twin..."
(Grieves, 2014)	"...contains three main parts: a)physical products in Real Space, b) virtual products in Virtual Space, and c) the connections of data and information that ties the virtual and real products together..."
(Grieves and Vickers, 2017)	"...is a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level..."
(Barricelli <i>et al.</i> , 2019)	"...can be defined as (physical and/or virtual) machines or computer-based models that are simulating, emulating, mirroring, or “twinning” the life of a physical entity, which may be an object, a process, a human, or a human-related feature..."
(Wu <i>et al.</i> , 2021)	"...is an intelligent and constantly evolving system, which monitors, controls, and optimizes the physical object through its lifecycle. The physical object may have many forms, such as a machine, a human, or a human-related thing, a smartcity. It should be noted that the data flow between a physical object and its virtual twin is bidirectional..."
(Mihai <i>et al.</i> , 2022)	"...can be more accurately thought of as a system-of-systems, a meld of several enabling technologies that construct an intelligent virtual representation of a physical entity and support a continuous two-way feedback loop between the twins..."

A recent study (Wu *et al.*, 2021) breaks down the definition of “Digital Twin” into three distinct categories, each focusing on a unique perspective and purpose.

- **The mirror model of a physical object** emphasizes the virtual representation of a physical entity but lacks the capability for automatic data exchange, resulting in a missing co-evolution between the physical and digital models.
- **The computerized simulation model** underscores the role of the computerized simulation in comprehending, predicting and optimizing the physical object. Specifically, it encompasses a unidirectional data flow from the physical entity to its digital reflection.

- **The integrated system** defines the digital twin as an intricate system consisting of the physical entity, its virtual counterpart, and the interconnectedness between them. This all-encompassing system continually aligns the virtual twin with operational modifications, enabling real-time predictions about the physical object's state.

The concept of the digital twin, from its initial conceptualization as a framework for [PLM](#) to its evolving definitions, exhibits the dynamic nature of this paradigm and its ability to incorporate advances in technology and expand its applications. Its emergence as a mirror model of a physical object, the evolution to a computerized simulation model, and finally, its manifestation as an integrated system, underline its potential to revolutionize the way we comprehend, predict, and optimize real-world systems. By synthesizing the physical and virtual realms, it facilitates real-time monitoring, operational adjustments, and predictive capabilities, thereby offering profound implications across a wide array of fields. As the concept continues to mature and be refined, it is evident that digital twin technology will remain at the forefront of intertwining our physical and virtual experiences, and its potential for future applications is boundless.

2.1.2 Enabling Technologies for Digital Twin

A digital twin is a system of systems that relies on multiple technologies to create a smart virtual replica of the corresponding physical entity and maintains a bidirectional communication link between the physical and virtual realms, supporting dynamic interactions and alignments ([Mihai et al., 2022](#)). The core technologies supporting digital twins include: Sensing and Measurement, 3D Modeling, [Machine Learning \(ML\)](#), [IoT](#), Distributed Computing, and [Human-Machine Interface \(HMI\)](#):

- **Sensing and Measurement:** Sensing and measurement technologies, such as cameras, [Light Detection and Ranging \(LiDAR\)](#), [Radio Detecting And Ranging \(RADAR\)](#), [Inertial Measurement Unit \(IMU\)](#), [Global Positioning System \(GPS\)](#), and other sensors and measurement devices, can capture various types of data, such as temperature, pressure, humidity, vibration, etc. ([Botín-Sanabria et al., 2022](#)).

Integrating sensor and measurement technologies into digital twins enhances the accuracy and realism of the virtual representation. The data collected through these sensors can be used to update and validate digital twins, ensuring that it reflects the current state of the physical system. Sensing and measurement data can also be used for predictive analytics and decision making within the digital twin systems.

- **3D Modeling:** 3D modeling serves as a powerful tool for forming virtual representations of physical assets and processes, enhancing the efficacy of digital twins through precise and comprehensive visual models (Qi *et al.*, 2021). These models embody real-world entities and offer capabilities for real-time observation, analysis, and simulation. First, 3D modeling allows for the fabrication of visually engaging and insightful models that portray physical assets and processes, offering users an interactive platform to monitor and assess real-world entities in real time. Second, through 3D modeling, digital twin systems can not only mimic, but also scrutinize the behavior and performance of the physical entities. With the integration of real-time sensor data and other inputs, these virtual models become predictive and prescriptive instruments, designed to optimize the functioning of physical assets. Furthermore, virtual models serve as experimental fields to test various design configurations, assess performance under different conditions, and pinpoint improvement opportunities. Additionally, 3D models simplify communication and collaboration amongst stakeholders, offering a universally comprehensible visual representation that bolsters decision-making and coordination.
- **Machine Learning:** ML technologies, encompassing traditional ML, DL, supervised, unsupervised, and Reinforcement Learning (RL), have found diverse applications within digital twin systems, as outlined in recent research (Mihai *et al.*, 2022). A prime example is the role ML plays in optimizing performance and supporting decisions. By processing and interpreting real-time data, ML can pinpoint areas ready for optimization, whether in energy efficiency, resource distribution, or process refinement. It also offers decision support by proposing action plans based on data analysis from both the physical system and its digital twin. The ability to sift through large amounts of data accrued from digital twin systems allows ML to uncover

critical insights and trends, enhancing our understanding of the performance and behavior of physical systems. The **ML** models, trained on historical data, can make informed predictions about future behavior or potential problems. These predictive capabilities enable digital twins to prevent and alleviate problems before their emergence, facilitating proactive maintenance and optimization. Furthermore, these **ML** models can be incrementally refined with freshly harvested data, continuously updated to increase their precision and efficacy. Through these **ML** functionalities, digital twins can gather information, generate predictions, optimize performance, and streamline decision-making processes.

- **Internet of Things:** The accelerated advancement of **IoT** infrastructures not only capacitates the digitization of billions of physical entities, but also underpins the seamless intercommunication between digital twins and their real-world counterparts (Wang *et al.*, 2023). At the microscopic level, **IoT** encompasses an intricate network of interconnected sensors, devices, and systems, which harvests real-time data from the physical environment. These **IoT** devices can provide a multitude of data, depending on the specific application. These data, collected by **IoT** devices, are relayed to digital twins, where they are processed, analyzed, and inform the construction of a virtual representation of the physical system. This virtual counterpart can accurately emulate the behavior, attributes, and performance of the physical system, in real-time. Digital twins persistently monitor and extract data from the physical system, enabling the generation of precise and current reflections. This continuous data exchange between the physical system and its digital twin facilitates unimpeded monitoring, analysis, and optimization of the physical system's performance. On a macroscopic level, digital twins can reside within cloud or edge servers, with a private synchronization link established for real-time data transfer between the digital twin and its physical entity, or other digital twins. This provision allows the digital twins to instantaneously visualize the status of its physical entity and to accelerate intelligent services such as 3D simulation, preventive maintenance, and informed decision-making. Furthermore, **IoT** undergirds the interconnection

of numerous digital twins, engendering **IoDT**. Within **IoDT**, an array of connected virtual twins can share information, harmonize statuses with physical objects, and perform cooperative tasks. This engenders an expansive information sharing network, laden with potential, where extensive distributed data, shared by various digital twins, can be effectively amalgamated to acquire comprehensive insights across a vast multitude of physical entities.

- **Distributed Computing:** Distributed computing encapsulates the coordination of multiple computers or devices that work together to execute tasks or solve problems (Mihai *et al.*, 2022). It enables efficient processing and analysis of voluminous data derived from both physical systems and their corresponding digital twins. Distributed computing facilitates the delegation of data processing and analysis tasks across a multitude of devices or nodes, eliminating the dependency on a centralized system. This results in faster and more effective data processing, with the added advantages of enhanced scalability and fault tolerance. In relation to the digital twin paradigm, distributed computing can facilitate intricate simulations, multivariate analyzes, **DL**-driven analyzes, and substantial data storage. Within such a distributed system, the cloud platform serves as a data repository, while the digital twin performs the synchronization of physical and virtual assets.
- **Human-Machine Interface:** **HMI** technologies, such as **AR** and **VR**, significantly enhance the interaction between human operators and digital twin systems by providing immersive and intuitive interfaces (Ma *et al.*, 2019). **AR** technology, which has the ability to overlay 3D digital graphics onto a real-world backdrop, effectively merges digital information into the physical environment. This fusion enables human operators to visualize and interact with digital twins in a more tangible and immersive manner. On the other hand, **VR** technology creates an enriched 3D scene that amplifies the **HMI** experience. Human operators can intimately engage with the digital world through wearable devices, such as mixed reality headsets, facilitating a more direct and authentic interaction with digital entities. This immersive interaction eliminates the barriers between virtual and real environments, fostering a seamless integration of both realms.

2.1.3 Digital Twin Applications

In the earlier stage, digital twin has gained recognition in astronautics and aerospace and has been adopted in NASA's technology roadmaps (Piascik *et al.*, 2010; Shafto *et al.*, 2012; Glaessgen and Stargel, 2012). According to the roadmap (Shafto *et al.*, 2012), NASA has designed four application scenarios around digital twins for its space mission, including:

- **Pre-flight Mission Simulation:** The digital twin can simulate the actual vehicle's future missions before its launch. It allows the effects of various mission parameters to be studied, and the impacts of possible anomalies are determining.
- **Mirroring Actual Flight:** Once the vehicle is in flight, the digital twin can be updated with actual load, temperature, and other environmental factors. This allows for continuous predictions for the flying vehicle.
- **In-situ Forensics:** The digital twin can be used to analyze potentially catastrophic events, as it closely mirrors the health state of the flying vehicle. It can diagnose the causes of any detected anomalies.
- **Mission Parameter Modification:** The digital twin serves as a platform where the effects of modifications to mission parameters, not considered during the design phase, can be studied.

The adoption of digital twin technology in space missions offers a host of considerable benefits, fostering enhanced mission preparedness and success. Leveraging its capability for preflight simulations, it facilitates a robust analysis of mission parameters and the prediction of potential anomalies, significantly mitigating risk. Through the real-time mirroring of the vehicle's flight, it enables continuous predictions and preventive maintenance, promoting operational longevity and reliability. The technology's capability for in-situ forensics allows for rapid anomaly diagnosis, ensuring timely responses to potential catastrophes. Finally, its role in studying modification of mission parameters facilitates flexible adaptation to unforeseen circumstances, thus strengthening the resilience of the mission and overall success rates. In essence, digital twin technology is an integral tool for enhancing the success, reliability, and efficiency of space missions.

Today, the usage of digital twin platforms has transcended its initial aerospace applications and has permeated diverse industries, highlighting its adaptability and the broad scope of its potential. Various digital twin applications have been developed for various industries (Javaid and Haleem, 2023; NVIDIA, 2023a).

- In the automotive industry, BMWGroup (2023) and Mercedes-BenzGroup (2023) harness digital twin technology to construct a virtual factory, integrating all existing tools and teams. This technological advancement helps to improve efficiency and optimize new production strategies, leading to substantial cost savings and improvement in production.
- In the energy sector, the U.K. Atomic Energy Authority (Chapman and Morris, 2019) has developed a full-scale digital twin of fusion reactors to enhance design and development efficiency. Siemens (2017) leverages digital twin technology to establish a unified digital grid model for the Finnish transmission system operator, Fingrid. This approach has yielded benefits such as reduced development time, superior data quality, standardized interfaces, and integration of grid models with asset management, leading to improved decision-making and operational efficiency.
- Within the infrastructure domain, Ericsson (2022) employs digital twin platforms to address a wide range of needs in network design, optimization, management and maintenance. Buildmedia (2021) has innovated a city-scale digital twin that consolidates open data from various sources for dynamic, real-time visualization, thus promoting collaboration between different stakeholders. HOK (2022) utilizes digital twin technology for the Centre Block rehabilitation project, incorporating virtual reality to facilitate exploration and interaction with a 3D representation of the Centre Block. Vouse (2021) implements a digital twin system for real-time monitoring and visualization of the position and movement of aircraft at Changi Airport.
- In the retail sector, Amazon Robotics (NVIDIA, 2023a) uses a digital twin system to generate photorealistic synthetic datasets that improve the performance of the neural network in autonomous warehouse robots and shorten development times for quicker iterative testing.

Looking ahead, the digital twin paradigm is poised to disrupt even more industries by enabling real-time monitoring, predictive analysis, and efficient system management. Integration of digital twins with emerging technologies such as [AI](#) and [IoT](#) could unlock unprecedented possibilities for system design, management and optimization. As we continue to innovate, the digital twin models will play an increasingly central role in the intersection of the physical and virtual world, making them an exciting field of study and application in the future.

2.1.4 Potentials and Challenges

The possibility of digital twin technology is already evident in the ability to replace expensive physical prototypes with modeling and simulation ([Grieves and Vickers, 2017](#)). Advancements in these areas, such as Boeing's "Atoms to Airplanes" ([Schneider, 2015](#)) approach for composite materials, showcase the potential of digital twins. Moreover, the inherent attributes of the digital twins lend it immense potential utility in various domains. One such application is the use of historical data accumulated by the system to streamline and optimize prospective system implementations and to preemptively identify undesirable yet unrealized behaviors. By digitally mirroring the system, designers can more effectively target system optimization and enhancements, concentrating their efforts on pivotal components. Concurrently, real-time "front-running" simulations offer the possibility of simulating the system's potential state during its operation. These "front-running" simulations of the system's possible 'future' enable operators to make optimal decisions. The efficacy of such anticipatory simulations is dependent on the pace with which the computational power exceeds the rate of physical activity. This novel approach marks a significant departure from traditional decision-making methods, enabling the system to predict the ramifications of a decision before its actual execution.

Despite the manifest potential of digital twin technology in a myriad of industries, several formidable challenges persist in multiple domains ([Botín-Sanabria *et al.*, 2022](#); [Singh *et al.*, 2021](#)).

- **Real-time data processing and synchronization:** A primary hurdle in digital twin implementation is the effective management of extensive data originating from diverse sources. These challenges are exacerbated by the heterogeneity of data sources, formats, and quality, demanding the establishment of standardized data structures to facilitate the progression of digital twin technology. Moreover, real-time data synchronization, a vital aspect that mirrors the state of physical entities and operational data within the virtual world instantaneously, presents a significant difficulty. Achieving a balance between synchronization rates and fidelity while managing massive amounts of data is not a trivial task.
- **Security and privacy:** The handling of sensitive information within digital twins requires rigorous measures for data security and privacy (Singh *et al.*, 2021). Cyber-attacks, data breaches, and unauthorized access can result in the exposure of sensitive information, making the implementation of encryption, access control, and security protocols an essential aspect of digital twin system development.
- **Interoperability and standardization:** The use of digital twins inevitably leads to the need for interoperability with various other systems. Despite the decade-long existence of the digital twin concept, standard data interface protocols are still conspicuously absent. This lack of a common standard hampers the effective exchange of data between disparate digital twin systems (Singh *et al.*, 2021). The introduction of an open standard for digital twins could ensure improved user experience and system compatibility.
- **Investment costs:** The design and deployment of digital twin systems is a costly endeavor that requires specialist knowledge of both hardware and software, as well as in-depth understanding of the modeled object (for instance, physical phenomena). To illustrate, modeling a wind farm necessitates understanding not only the farm itself but also the software system running on the hardware and the intricacies of wind dynamics. On the other hand, implementing and operating the digital twin required large amounts of computational resources (Gabor *et al.*, 2016). On the basis of the simulation level and scale, the costs associated with developing digital twin systems that accurately respond to physical behavior can be substantial.

- **Understanding and modeling of the physical world:** Despite considerable advancements in 3D modeling techniques, creating detailed and precise models that accurately mirror complex physical systems remains a formidable challenge. The construction of a digital twin requires an accurate simulation of complex systems and natural phenomena (Grieves and Vickers, 2017). However, human comprehension of these phenomena is still largely limited, invariably impeding the fidelity of digital twin systems. In addition, complications in physical systems, data scarcity, and the need for specialized expertise in modeling and simulation add further layers of complexity. Potential approaches to surmount these challenges may involve the application of data-driven techniques like ML and DL for constructing and calibrating 3D models. Additionally, it is imperative to bolster our understanding and modeling capabilities of materials, structures, mechanics, and microphysical phenomena to yield more insightful and meaningful digital twin analyses.

2.2 AI-Enabled Wireless Systems

An AI-enabled wireless system integrates AI technologies into wireless communication networks, enhancing their performance, efficiency, adaptability, functionality, and user experience. AI technologies are applied at multiple levels in wireless networks, encompassing signal processing, network management, user interaction, and the expansion of features. In today's world, the advent of wireless sensor technology and the IoT has enabled the seamless interconnection of an extensive array of sensors, intelligent devices, software, hardware systems, vehicles, and even smart buildings (Liu and Wang, 2019). This complex web forms a comprehensive, wirelessly connected information network. The integration of AI into wireless systems has endowed these networks with unprecedented intelligence, leading to the creation of various intelligent wireless applications. These applications span across several domains, including smart cities, autonomous vehicles, intelligent home appliances, smart agriculture, and automated retail sectors.

2.2.1 Evolution: From Maxwell's Equation to Smart Wireless Systems

In 1873, **Maxwell (1873)** introduced a seminal piece of work, "A Treatise on Electricity and Magnetism", delineating the propagation of **Electromagnetic (EM)** waves in space through intricate mathematical formalism. Here is the integral form of Maxwell's equations.

$$\begin{aligned}
 \oiint \mathbf{E} \cdot d\mathbf{S} &= \frac{1}{\varepsilon_0} \iiint \rho dV \\
 \oiint \mathbf{B} \cdot d\mathbf{S} &= 0 \\
 \oint \mathbf{E} \cdot d\boldsymbol{\ell} &= -\frac{d}{dt} \iint \mathbf{B} \cdot d\mathbf{S} \\
 \oint \mathbf{B} \cdot d\boldsymbol{\ell} &= \mu_0 \left(\iint \mathbf{J} \cdot d\mathbf{S} + \varepsilon_0 \frac{d}{dt} \iint \mathbf{E} \cdot d\mathbf{S} \right)
 \end{aligned} \tag{2.1}$$

Here $\mathbf{E}$ is the electric field. $\mathbf{B}$ is the magnetic field. $\mathbf{J}$ is the current density. $d\mathbf{S}$ is the differential area element of the closed surface. dV is the differential volume element. $d\boldsymbol{\ell}$ is the differential element of length along the path. ρ is the charge density. ε_0 and μ_0 are the permittivity and permeability of free space. The Maxwell's equations demonstrate the interrelated nature of electric and magnetic fields and are central to understanding electromagnetic phenomena, including light, radio waves, and electricity. They form the foundation of many modern technologies and are a theoretical and applied physics cornerstone. The propagation of the **EM** field can be described by differential forms.

$$\begin{aligned}
 \nabla \cdot \mathbf{E} &= \frac{\rho}{\varepsilon_0} \\
 \nabla \cdot \mathbf{B} &= 0 \\
 \nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\
 \nabla \times \mathbf{B} &= \mu_0 \left(\mathbf{J} + \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} \right)
 \end{aligned} \tag{2.2}$$

The Maxwell equations and their corresponding differential representations formulate four fundamental principles in the realm of electromagnetism.

- **Gauss's Law for Electric Fields**, delineating the manner in which electric charges give rise to electric fields.
- **Gauss's Law for Magnetic Fields**, postulating the nonexistence of magnetic monopoles, implying magnetic fields consistently establish closed loops.
- **Faraday's Law of Electromagnetic Induction**, explicating the phenomenon by which a variable magnetic field engenders an electric field.
- **Ampere's Law, supplemented with Maxwell's Addition**, demonstrating that magnetic fields are elicited by both electrical currents and altering electric fields, the latter being Maxwell's crucial amendment that rectified the inconsistencies in Ampere's original law, rendering it congruent with the principle of charge conservation.

These laws form the underpinnings of contemporary physics and technology, spanning disciplines and applications such as optics, electrical circuits, and radio waves, among many others. These equations collectively offer a comprehensive description of classical electromagnetism, encapsulating the behaviors and interplay of electric and magnetic fields.

Building on Maxwell's foundational research, Hertz created the inaugural prototype for wireless communication devices in 1888. This pioneering device could generate, detect and receive EM signals (Hempstead and Worthington, 2005). Subsequently, thanks to the relentless endeavors of scientists, the "spark-gap" transmitter emerged as one of the earliest radio transmitters and was practically employed across diverse applications, including military and maritime use. It was not until 1912, with the advent of the triode vacuum tube capable of producing high-quality sinusoidal EM waves, that the harder-to-modulate "spark-gap" transmitters began to be replaced.

In the early days of electrical communications, communication relied on wired connections between transceivers. Radio communications as a revolutionary method freed communication from the constraints of physical connections. This form of communication was called "Wireless Communication" in the early days and is still used today. The semantics of "Wireless"

has now been further expanded and is commonly used to denote the services provided by communication technologies that rely on EM waves propagating in space. This phenomenon reflects the shift in the dominant mode of communication from a wired connection to a wireless format.

In the contemporary world, cellular networks have emerged as the most prevalent form of wireless communication. By 2023, billions of people worldwide had access to cellular services (Taylor, 2023). The inception of the cellular network can be traced back to its invention in the late 20th century at Bell Labs in the USA (Anderson, 2003). Official deployment began in the late 1970s to early 1980s. Currently, Fifth Generation (5G) voice and data communication technologies are integral to everyday life, and researchers are already working toward Sixth Generation (6G).

In the modern era, the emergence of wireless sensor technology and the IoT has facilitated the interconnection of a vast number of sensors, intelligent devices, software, hardware systems, vehicles, and even smart buildings (Liu and Wang, 2019). This intricate interconnected fabric forms a comprehensive information network that is wirelessly linked. Furthermore, the incorporation of AI technology into wireless systems has provided remarkable intelligence to the network. This development has spawned a range of intelligent wireless applications, including smart cities, autonomous vehicles, intelligent home appliances, smart agriculture, and automated retail.

As depicted in Figure 2.3, an example of a wireless system powered by AI is the wireless network system itself (Elsayed and Erol-Kantarci, 2019). Integration of AI algorithms can help reduce network latency and improve system stability. Moreover, DL algorithms can be used to improve resource allocation in communication, allowing the network system to adaptively adjust the distribution of resources based on usage patterns, thereby achieving optimal resource scheduling.

Another typical application is the AI-based wireless location and tracking system, as shown in Figure 2.4. By utilizing AI algorithms and wireless sensor measurements, this system can accurately determine the position of smart devices.

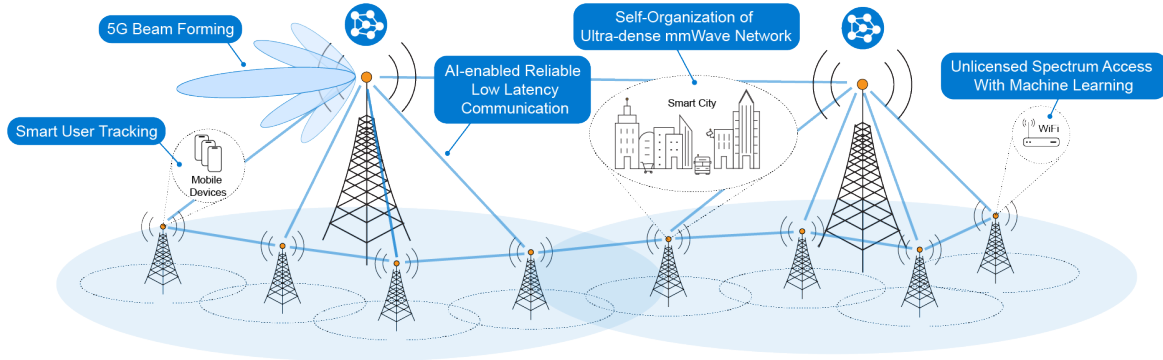


FIGURE 2.3: AI-enabled wireless network system.

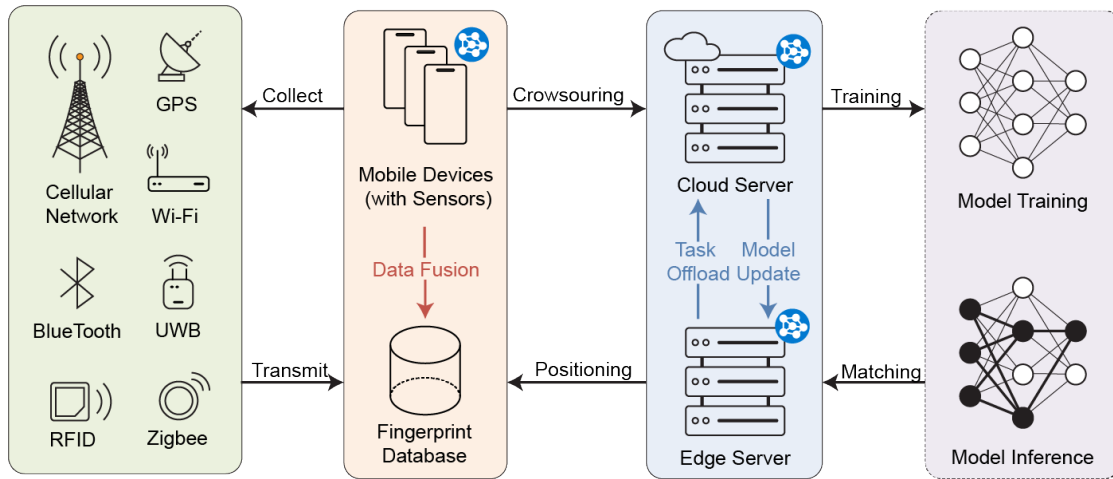


FIGURE 2.4: An AI-enabled wireless positioning system.

Furthermore, human activities can exert an influence on the propagation of wireless signals. Analyzing the response of these signals to human activity can glean a wealth of information about such behaviors. This information not only deepens our understanding and analysis of human behavior, but also offers critical insights for enhancing the performance of the wireless system itself.

2.2.2 AI-Enabled Wireless Systems for Indoor Positioning

Wireless communication and IoT technologies have contributed significantly to the advancement of wireless indoor positioning systems. These systems utilize sensor data to estimate the location of devices or users in indoor environments. However, indoor positioning presents

unique challenges compared to outdoor positioning, as it involves complex building structures and dynamic interference caused by human activities. These factors hinder the acquisition of accurate sensor measurements, leading to difficulties in achieving precise positioning results. Despite extensive research and development efforts, a wireless positioning system that encompasses affordability, stability, accuracy, usability, and scalability remains elusive. Researchers persistently seek a balance between complexity and accuracy in their search for an optimal solution.

Over the years, researchers have proposed various techniques and algorithms for indoor positioning. Techniques such as Bluetooth (Feldmann *et al.*, 2003; Wang *et al.*, 2013), WiFi (Yang and Shao, 2015; Shchekotov, 2014; He and Chan, 2015), Ultra-wideband (UWB) (Alarifi *et al.*, 2016), Zonal Intercommunication Global-standard (ZigBee) (Huang and Chan, 2011), Radio Frequency Identification (RFID) (Seco *et al.*, 2010), infrared (Liu *et al.*, 2007), Frequency Modulation (FM) (Xiao *et al.*, 2016), ultrasound (Morohin and Totsky, 2021) and acoustic signals (Sertatil *et al.*, 2012) have been explored. Additionally, algorithms that include triangulation (Wang *et al.*, 2013), trilateration (Shchekotov, 2014), fingerprinting (He and Chan, 2015), ML (Guo *et al.*, 2019) and DL (Ayyalasomayajula *et al.*, 2020) have been developed to enable high-precision positioning.

The widespread adoption of smart devices and smartphones has made positioning technology more accessible to the general public. Location information obtained from wireless positioning systems holds significant value in diverse industries such as transportation, healthcare, education, emergency response, construction, and logistics. Moreover, wireless Indoor Positioning System (IPS) play a pivotal role in the realization of large-scale intelligent systems, including smart cities, smart buildings, and digital twins. These systems offer substantial potential to improve efficiency and enable advanced functionalities in various domains.

2.2.2.1 Techniques and Technologies

In simple terms, the basic principle of wireless indoor positioning is to map physical measurements from wireless signals to geometric information such as distance and direction, or to

predefined coordinates (Yang *et al.*, 2013). These physical measurements can be divided into four main categories, power, time, angle, Channel State Information (CSI).

- **Power:** Signal power, due to its inherent accessibility in wireless devices, frequently underpins indoor positioning applications. Received Signal Strength Indicator (RSSI), a prevalent power metric, is incorporated into various wireless technologies, including Bluetooth (Zhou *et al.*, 2017), WiFi (Liu *et al.*, 2020), and UWB (Wang *et al.*, 2010). However, the accuracy of signal power measurement in indoor positioning is limited due to its susceptibility to interference caused by complex indoor environments with multiple propagation paths and human activity. Consequently, positioning methods predicated on signal power often lack substantial accuracy. Alleviating interference induced by multipath effects requires a precise model of these effects; however, creating such a model is typically a challenging task (Zayets and Steinbach, 2017).

- **Received Signal Strength (RSS):** RSS is the most commonly employed metric in IPSs, due to its simple acquisition. This method estimates the distance between the receiver and the transmitter by gauging the intensity of the signal received in decibel milliwatts (dBm). As the point source electromagnetic field propagates through space, it attenuates with increasing distance, which causes a decrease in measured RSS. Under ideal Line-of-Sight (LoS) conditions, this relationship between signal strength and distance attenuation can be expressed as:

$$P_r(d) = P_t G_t G_r \left(\frac{\lambda}{4\pi d} \right)^2 \quad (2.3)$$

where $P_r(d)$ denotes the received power, P_t signifies the transmitted power, d represents the distance between the transmitter and receiver, G_t and G_r correspond to the antenna gain of the transmitted and received antennas, respectively, and λ is the wavelength of the transmitted signal. In practice, **Path Loss Models** (Neskovik *et al.*, 2000) are commonly used to model the relationship between RSS and distance, which takes the following general form:

$$\mathcal{L}_{\text{path}}(d) = \mathcal{L}_{\text{path}}(d_0) + 10\eta \log\left(\frac{d}{d_0}\right), \forall d \geq d_0 \quad (2.4)$$

Here, $\mathcal{L}_{\text{path}}(d)$ represents the path loss in dB at distance d . $\mathcal{L}_{\text{path}}(d_0)$ signifies the power loss at a reference distance, where d_0 is typically set at 1 meter, containing all constant factors such as transmit power and system gains. The path loss exponent η describes the attenuation effect and usually falls between 2-5 meters.

- **CSI:** In contrast to **RSS** measurements, **CSI** estimates both amplitude and phase measurements for each subcarrier (Yang *et al.*, 2013). This feature enables it to discern multipath effects, thus exceeding the constraints of **RSS**. The **Channel Impulse Response (CIR)**, a type of **CSI** measurement, functions as a time-linear filter for the wireless propagation channel. Assuming time invariance, it can be formulated as follows:

$$h(\tau) = \sum_{i=1}^N a_i e^{-j\theta_i} \delta(\tau - \tau_i)$$

where a_i denotes the amplitude, θ_i represents the phase, τ_i is the time delay of the i^{th} path, N is the total number of multipath components, and $\delta(\tau)$ signifies the Dirac delta function. Each term in the sum corresponds to an individual delayed multipath component. In the frequency domain, constructive and destructive phases lead to frequency selective fading, which is characteristic of **Channel Frequency Response (CFR)**. **CFRs**, essentially the Fourier transforms of **CIRs**, describe small-scale multipath effects and are extensively used in channel measurements. Compared to **RSS**, channel measurements provide finer spatial and temporal resolution, leading to a significant improvement in measurement accuracy.

- **Time:** **Time of Arrival (ToA)**, **Time Difference of Arrival (TDoA)**, **Round Trip Time of Arrival (RTT)** techniques are frequently employed in wireless signals for range estimation (Kaune, 2012). The accuracy of temporal range can be enhanced by increasing the signal bandwidth. In particular, the **UWB** and high temporal resolution of **UWB** technologies enable distance measurements with centimeter-level precision

(Farahsari *et al.*, 2022). Although time measurements exhibit exceptional accuracy under LoS conditions, frequent occlusions in indoor environments often result in Non-Line-of-Sight (NLoS) scenarios. Furthermore, time-based ranging techniques require an external signal source and are invariably linked to substantial energy consumption (Yang *et al.*, 2013).

- **ToA** method estimates distance by calculating the propagation time of the signal between transceivers. This technique generally requires a time synchronization mechanism to ensure accuracy. The effectiveness of its positioning is bounded by the device's bandwidth and sampling rate.
- **TDoA**: TDoA, an improvement over ToA, only requires synchronization of time between the anchor points. TDoA estimates the distance by computing the time difference between the propagation signal and each anchor point. For successful localization using this method, the target must be within the detection coverage area of at least three anchor points. The accuracy of TDoA is also limited by the sampling frequency, bandwidth, and whether the signal propagation is under LoS conditions.
- **RTT** estimates the distance by measuring the time it takes for a signal to propagate from the transmitter to the receiver and back. Unlike ToA and TDoA, the accuracy of RTT is not dependent on time synchronization. However, a precise timer is necessary for the calculation.
- **Angle**: Angle information, such as Angle of Arrival (AoA) (Zheng *et al.*, 2018), complements distance measurements in indoor positioning, significantly enhancing positioning accuracy. By combining distance and angle measurements, positioning can be achieved using a single anchor point (Yang *et al.*, 2013). However, the detection of angles requires the use of antenna arrays, leading to a higher cost of angle measurement equipment compared to distance measurement devices.
 - **AoA** determines the AoA of a signal by detecting signals from multiple directions. In the early days, AoA used a rotating directional antenna to measure the direction of the strongest signal strength to determine the angle. However, such rotating directional antennas are not commonly used and are difficult to use in

a wide range of positioning scenarios. With the popularity of **Multiple-Input Multiple-Output (MIMO)** technology in standard **Wireless Local Area Network (WLAN)** protocols, it is more feasible to estimate **AoA** by measuring **CIRs** from an antenna array.

Many positioning techniques have been implemented and used in various wireless indoor positioning systems. These technologies include, but are not limited to, Bluetooth, WiFi, **UWB**, **RFID** and many more. The Table 2.2 shows the performance and characteristics of the different positioning technologies.

TABLE 2.2: Indoor Localization Technologies

Technologies	Techniques	Accuracy	Coverage	Robustness	Energy	Cost	Literature
UWB	RSS, TDoA, ToA, RTT	cm-m	<200m	High	Medium	High	(Alarifi <i>et al.</i> , 2016)
Bluetooth	RSS, AoA	cm-m	<400m	Medium	Low	Low	(Wang <i>et al.</i> , 2013)
Wi-Fi	RSS, AoA, CSI, RTT	m	<100m	Medium	High	Low	(Yang and Shao, 2015)
Zigbee	RSS	m	<100m	Poor	Low	Low	(Fang <i>et al.</i> , 2012)
RFID	RSS	m	<100m	Good	Low	Low	(Saab and Nakad, 2010)
VLC	LiFi	cm-m	<60m	Poor	High	High	(Lam and Little, 2019)
Acoustic	ToA, TDoA	cm-m		Poor	Low	Low	(Höflinger <i>et al.</i> , 2012)
Ultrasonic	ToA, TDoA	cm-m		Poor	Low	High	(Hazas and Hopper, 2006)
IMUs		m		Poor	Low	Low	(Feng <i>et al.</i> , 2020)
GF		m		Medium	Low	Low	(Bae and Choi, 2019)

- **UWB** transmits data across a wide bandwidth using ultra-narrow pulses in the nanosecond range (<1 ns). Operating in the 3.1-10.6 GHz band, **UWB** boasts a low duty cycle, which facilitates operation at reduced power consumption. Taking advantage of ultra-bandwidth, **UWB** provides high spatial and temporal resolution for the transmitted signal. This enables effective segregation of the signal's multipath component and the identification of the direct transmission component, thus achieving centimeter-level accuracy. With its high precision, low power consumption, strong penetration capability and low complexity, **UWB** is widely adopted in indoor positioning. Here, the distance between the tag and the anchor (base station) is typically estimated using the methodologies **ToA** (Kok *et al.*, 2015), **TDoA** (Cheng and Zhou, 2019), and **RTT** (De Angelis *et al.*, 2016).
- **Bluetooth** is a widely adopted technology for indoor positioning due to its low cost and energy consumption. It operates in the 2.4 GHz band, following the IEEE 802.15.1 standard, and in its latest version it can cover a range of up to 400 meters.

Bluetooth can support **RSS** (Wang *et al.*, 2013), **AoA** (Toasa *et al.*, 2021), and **ToA** (Geng *et al.*, 2021) methodologies, but **RSS** is predominantly used due to the simplicity of its implementation and excellent compatibility (e.g., with Apple and Android smartphones). iBeacon (Apple, 2013) and Eddystone (Google, 2015) are two protocols based on **Bluetooth Low Energy (BLE)** proposed by Apple and Google, respectively, mainly used for proximity detection. Generally, the use of Bluetooth for indoor positioning hinges on fingerprint algorithms, in which fingerprint features in a given scene are gathered for location matching.

- **Wi-Fi** (IEEE 802.11 standard) is used in a wide range of environments, from airports and supermarkets to hospitals and private homes. Originally designed to offer wireless communication support for mobile devices, Wi-Fi has emerged as a key player in indoor positioning scenarios. Technologies such as **RSS**, **ToA**, **AoA**, and **CSI** are instrumental in Wi-Fi-based positioning systems (Yang and Shao, 2015; Liu *et al.*, 2020). Despite potential limitations in positioning accuracy due to complex indoor environments and significant interference from human activity, Wi-Fi remains the preferred solution due to its ubiquity, expansive coverage, high data rate, and broad compatibility. Although **CSI** can offer centimeter-level accuracy, its implementation requires expensive and dedicated facilities. Therefore, the **RSS** information of Wi-Fi, similar to Bluetooth, is more competitive in real-world applications. Various studies suggest that the accuracy and stability of Wi-Fi-based positioning systems can be significantly improved using methods such as crowdsourcing and fingerprint calibration (Wang *et al.*, 2016; Jang and Kim, 2018).
- **ZigBee** a communication technology grounded in the IEEE 802.15.4 standard, is characterized by its low cost, reduced data rate and minimal power consumption, making it an optimal choice for applications in smart homes and wireless sensor networks. Due to its **RSS** mirroring those of Wi-Fi and Bluetooth, **ZigBee** is frequently incorporated in indoor positioning solutions, offering a coverage range of up to 100 meters and providing meter-level positioning accuracy, which is suitable for buildings and subterranean spaces. Although **ZigBee** is widely used for sensor positioning within wireless sensor networks, the lack of compatibility with common

user devices such as smartphones poses a challenge for its implementation as a user positioning solution (Zafari *et al.*, 2019).

- **RFID** a technology created for data storage and retrieval, is frequently employed in industries such as retail and warehousing for the tracking of physical assets. The accuracy of **RFID** localization depends on factors such as device coverage, antenna types, and signal frequency. Generally, **RFID** systems are made up of three key components: readers, tags, and servers. Depending on their functionality, these tags can be classified into three categories: active, passive, or semi-passive:
 - Active **RFID** tags, powered by an internal battery, can function over long distances. The communication range between the active **RFID** tag and its reader can extend from tens to hundreds of meters. However, the inclusion of an internal battery increases the size and cost of the tag, while limiting its service life.
 - Passive **RFID** tags do not contain a battery; instead, they derive power from the **RFID** reader. When in proximity to the **RFID** reader, the antenna of the passive **RFID** tag antenna transforms the electromagnetic wave energy received into electrical energy. This activates the chip within the **RFID** tag, triggering the transmission of the data stored on the chip. Free from the limitations of battery dependence, passive **RFID** tags offer advantages such as small size, light weight, low cost, and long service life. They can be fabricated into various forms, such as thin sheets or hanging tags, tailored to specific applications and environments. However, the communication range between passive **RFID** tags and readers is generally limited to a few meters and often requires a high-power **RFID** reader.
 - Semi-passive **RFID** tags blend the advantages of active and passive **RFID** tags. These tags remain dormant and do not transmit **RFID** signals unless they enter the activation signal range of a low-frequency activator, at which point they become active. Although the semi-passive **RFID** tags contain a built-in battery, they are only used to power the internal circuitry and do not contribute to signal transmission. The energy transfer in semi-passive **RFID** is similar to that in

passive systems, resulting in a significantly longer operational life compared to traditional active system tags.

- **Visible Light Communication (VLC)** is a high-speed data transmission technology that leverages **Light Emitting Diodes (LEDs)** as its signal source. It uses a visible light source to emit a pulsating light signal. This signal is then received by a smartphone or other optical sensor for distance estimation. **VLC** stands out for its immunity to interference from other signals and its significant suppression of multipath effects. However, due to the low penetration of visible light, **VLC** requires **LoS** conditions to deliver effective positioning.

In addition to these electromagnetic wave-based technologies, acoustic, ultrasonic, inertial navigation, and geomagnetic sensors are also being investigated for indoor positioning.

- **Acoustic** technology uses modulated sound signals to carry time-dependent information, which are then captured by a smartphone microphone to enable **ToA** calculations. The Doppler effect, which induces changes in the sound phase and frequency as a result of relative motion, can be utilized to estimate the relative speed and position of devices. Despite the demonstrated high accuracy of acoustic technology, its application remains limited due to the necessity for dedicated facilities and its high power consumption.
- **Ultrasonic** (frequencies >20KHz) positioning technology determines relative distances through the **ToA** of sound propagation between transceivers. This technique offers accuracy at the centimeter level and can track several moving targets simultaneously. Its significant penetration ability serves as a crucial advantage, rendering it highly effective in indoor settings where obstructions like walls, doors, and windows do not hinder its performance. However, ultrasonics is sensitive to environmental factors such as temperature and humidity. To mitigate these effects, it is typically combined with temperature and humidity sensors, along with specific algorithms designed to suppress environmental influences.
- **IMU** generally comprise a 3D accelerometer and a 3D gyroscope, together offering six dimensions of measurement data. Through continuous tracking of these inertial

navigation sensors, the device's velocity and displacement can be inferred. Modern smartphones often come equipped with built-in **IMUs**, independent of external systems. This allows their integration with other positioning systems such as **BLE** (Feng *et al.*, 2020) and Wi-Fi (Huang *et al.*, 2023), thus facilitating hybrid positioning solutions. However, **IMUs** are subject to error accumulation, a problem commonly mitigated using Kalman algorithms, particle filtering, and other similar methods.

- **Geomagnetic Field** characterized by high spatial variability and excellent temporal stability, is often used as a location feature in the construction of fingerprint libraries for **IPSs**. The complexity of reinforced concrete structures in buildings improves the textural nuances of the indoor geomagnetic field, thereby increasing the spatial variability of geomagnetic signals and strengthening its suitability for **IPSs**. Modern smartphones, equipped with built-in geomagnetic sensors, integrate seamlessly with geomagnetic-based positioning systems. Various studies have attested to the potential to achieve meter-level positioning accuracy when using geomagnetism as a fingerprint feature (Bae and Choi, 2019).

2.2.2.2 Positioning Approaches

The main idea behind positioning is the conversion of sensor measurements into location information, and the main approaches can be divided into two broad categories: geometric mapping and fingerprinting (Yang *et al.*, 2013).

Geometric Mapping

Geometric mapping translates acquired sensor signals into geometric measurements of distance and direction. It then converts the geometric relationship between the localized target and the anchor nodes into the target's position. Distance-based geometric mapping approaches are more common, as the parameters needed are simpler to acquire compared to those required for angle-based methods. To measure distance, methods based on **RSS** are popular due to their compatibility and accessibility. Distance estimation can also be performed using time-based methods **ToA**, **TDoA** and **RTT**, as described in Section 2.2.2.1.

Lateration and Angulation are the two primary algorithms used for geometric mapping:

- **Lateration:** This technique estimates the position by measuring the distance between the target and three or more anchor nodes. It requires the target to be within the common coverage area of the signals from at least three anchor nodes, and the localization accuracy increases with the number of anchor nodes and measurements.
- **Angulation:** Angulation leverages the [AoA](#) to derive the relative angles between the target and the anchor node with known distances between these anchor nodes for position estimation. Much like Lateration, the accuracy of angulation is associated with the number and placement of anchor nodes, as well as the number of measurements taken. However, the effectiveness of angulation is closely dependent on [LoS](#) conditions, making it less useful in complex indoor positioning scenarios. The algorithm for angulation is more intricate than that of lateration. For a more detailed explanation of this method, refer to ([Esteves *et al.*, 2003](#)).

Although geometric mapping approaches are widely used because of their simplicity and low computational complexity, their accuracy is limited by the lack of detailed descriptions of complex indoor environments.

Fingerprinting

Fingerprinting estimates the position using location-based measurements as features, which are compared with a constructed fingerprint library. As previously discussed, the reduced accuracy of lateration and angulation is largely due to the mathematical models they employ, which do not adequately represent complex indoor environments. Fingerprinting bypasses the need to map intermediate quantities, such as distances and angles, thereby reducing its dependency on precise mathematical modeling.

Moreover, fingerprinting leverages location-based measurements as features to construct a fingerprint library, making it highly compatible with various types of measurements. Measurements based on power, angle, time, and geomagnetic information can be integrated as fingerprint features. Given its high feature dimensionality and independence from the accuracy of mathematical modeling, fingerprinting exhibits superior performance compared to

geometric mapping-based approaches. Furthermore, the data used as features, such as [RSS](#) and geomagnetic information, are readily available in practical scenarios and can be used with existing infrastructure. This significantly reduces the costs associated with localization.

The fingerprint-based location framework typically involves two stages: an offline phase and an online phase.

- The **offline phase** is dedicated to constructing and calibrating the fingerprint library. During this stage, the user moves throughout the environment with a measuring device (such as a smartphone or laptop), records sensor readings, and calibrates the position, whether stationary or in motion.
- In the **online phase**, the user's device transmits real-time signal data to the location engine. This engine is responsible for matching these fingerprints and estimating the device's location. Once the location is determined, the engine returns the results to the user's device.

Although fingerprinting has distinct advantages over geometric mapping, this technique has two major limitations:

- The construction of a fingerprint database is a time-consuming and expensive process. The quality of the fingerprint database is critical in determining the position accuracy of the [IPSSs](#). Sensor readings from different devices (e.g., varying brands of smartphones) can significantly differ at the same location, leading to substantial discrepancies in positioning accuracy across different users' devices. A high-quality fingerprint library must also consider compatibility with various user devices. Moreover, a too dense collection of fingerprints can hinder the speed of localization, while a collection that is too sparse can compromise the accuracy of the localization. Some studies propose the use of crowd-sourcing to alleviate the burden of fingerprint capture, a method already implemented in certain commercial software solutions ([Ni et al., 2022](#)).
- In reality, signal distributions across space vary over time, leading to time-sensitive fingerprints. In the same environment, human activity (such as movement) can cause

changes in signal distribution, which may result in inconsistent fingerprint features captured at the same location, thereby causing localization errors. Additionally, movements of signal sources, such as WiFi devices, changes in office equipment placement, or inventory change can lead to alterations in wireless signal distribution. Under such circumstances, the fingerprinting results can show substantial variations. Consequently, regular updates to the fingerprint database are crucial to ensure the accuracy of the location database.

2.2.2.3 Algorithm for Wireless Positioning

The algorithms to estimate the position in [IPSS](#) can be categorized as conventional methods and machine learning approaches.

Conventional Methods

A survey ([Shang et al., 2015](#)) divided conventional methods into Bayesian and non-Bayesian methods. Compared to non-Bayesian methods, Bayesian methods treat the target position as a random variable x with a prior distribution $p_x(x)$, rather than unknown parameters. Four common conventional methods were introduced by another survey ([Guo et al., 2019](#)):

- **Least Squares:** Least squares methods are commonly used to solve positioning problem with redundancy observations from measurements (e.g, [RSS](#), [ToA](#), [TDoA](#)). The main idea of least squares is to minimize the residuals to yield an approximate solution. A least squares-based positioning problem can be expressed as:

$$\hat{\mathbf{z}} = \arg \min_{\mathbf{z}} \|\mathbf{s} - g(\mathbf{z})\| \quad (2.5)$$

Here $\mathbf{z}$ represents the estimated location, $\mathbf{s}$ denotes the measurements. $g(\mathbf{z})$ is the geometric mapping function that converts measurements to location information.

- **Maximum Likelihood:** Given the conditional [Probability Density Function \(PDF\)](#) $p_{\mathbf{s}|\mathbf{z}}$ of the observation, the maximum likelihood technique estimates the position $\mathbf{z}$

by optimizing the following likelihood function:

$$\hat{\mathbf{z}} = \arg \max_{\mathbf{z}} p_{\mathbf{s}|\mathbf{z}}(\mathbf{s}|\mathbf{z}) \quad (2.6)$$

Typically, $p_{\mathbf{s}|\mathbf{z}}$ is approximated using a parametric distribution such as multidimensional Gaussian distribution.

- **Maximum A Posteriori (MAP):** The **MAP** approach solves the positioning problem by maximizing the posterior probability, which can also be viewed as a regularized form of maximum likelihood:

$$\hat{\mathbf{z}} = \arg \max_{\mathbf{z}} p_{\mathbf{z}|\mathbf{s}}(\mathbf{z}|\mathbf{s}) \quad (2.7)$$

In practice, given the observation of measurements, the probability distribution of the position $\mathbf{z}$ can be deduced using Bayesian criteria. For example, the path loss model describes the relationship between **RSS** and distance. The posterior $p_{\mathbf{z}|\mathbf{s}}$ can be approximated by integrating the path loss model with a Gaussian distribution. Consequently, the estimation is accomplished by maximizing the probability derived from the combined density distribution functions of the observations.

- **Minimum Mean Square Error (MMSE):** This approach estimates position by minimizing the average of the positioning errors:

$$\hat{\mathbf{z}} = \int \mathbf{z} p_{\mathbf{z}|\mathbf{s}}(\mathbf{z}|\mathbf{s}) d\mathbf{z} \quad (2.8)$$

MMSE methods can be used to train or calibrate the **ML** model describing $p_{\mathbf{z}|\mathbf{s}}$. The location can then be estimated by giving the observation as input using $p_{\mathbf{z}|\mathbf{s}}$.

Machine Learning Methods

In indoor positioning, **ML** methods can be divided into classification and regression. The essence of such algorithms is to train **mML** models using the collected measurements and the corresponding location labels, and then use the trained models for location estimation. This type of approach can be expressed as

$$\hat{y} = f(\mathbf{s}, \mathcal{D}) \quad (2.9)$$

where $\mathcal{D}$ is the offline training data, s is the online measurement, $f(\cdot)$ is the **ML** model, $\hat{y}$ is the training label. Here, four popular **ML** approaches are introduced:

- **K-Nearest Neighbor (KNN):** KNN algorithms estimate position by averaging the k -nearest matches known position using online measurement. In KNN-based fingerprinting, the first step is defined by a distance metric to calculate the measurement distance from each fingerprint in the database. Then, the corresponding position of top- k -nearest matches record will be found and averaged as the final estimation. A typical KNN algorithm can be written as follow:

$$\hat{\mathbf{z}} = \frac{1}{k} \sum_{i=0}^k \mathbf{z}_i, \mathbf{z}_i \in h(\mathbf{s}, k) \quad (2.10)$$

where $h(\cdot)$ is a function that returns a set containing the top- k nearest matched position according to given k and measurements $\mathbf{s}$.

- **DL:** DL is the most popular area of **ML**. His excellent algorithmic performance has been widely recognized in many fields. DL trains Deep Neural Networks (DNNs) by means of a backpropagation algorithm. Through a multilayer neural network architecture, DL models can effectively learn potential connections in data and apply them to various fields. In the field of localization, some research (Abbas *et al.*, 2019; Xiao *et al.*, 2017) has been conducted to suppress noise in the received signal by DL methods. Other studies (Adege *et al.*, 2018; Lee *et al.*, 2018), use DL as a classifier to directly engage in localization.

2.3 Wireless Signal Simulation

Wireless signal simulation is an essential aspect of modern telecommunications and network planning. It does so by constructing virtual models of the wireless communication environment to analyze and predict the behavior of radio waves as they propagate through various media. This process is critical for optimizing the performance of wireless networks such as cellular, Wi-Fi, and satellite communications. In general, simulations consider various factors such as frequency, power level, antenna type, and the physical environment, including

buildings, trees, and other obstacles. Engineers and network designers can predict signal strength, coverage, and potential interference problems by simulating these conditions. This predictive capability can improve network design efficiency, reduce costs, and improve quality of service.

2.3.1 The Role of Simulation

Simulation is the replication of real-world phenomena or operations over time. It enables researchers to construct models that allow for an in-depth examination of system behaviors under various conditions. Before actual system implementation, simulation facilitates predictive insights into the system's performance within real-world scenarios.

The sophistication of intelligent wireless systems has led to a wide application of simulation within this field. The complexities inherent in such systems stem not only from their intricate internal architecture, but also from the diverse and challenging environments within which they are deployed (Tranter, 2004). From the system's perspective, wireless communication technologies are tasked with the provision of high data rates while concurrently managing power usage and bandwidth. These demands result in complex signal processing and system implementations. Furthermore, as wireless systems depend on the transmission of electromagnetic waves for communication, the complexity of the deployment environment (e.g., multipath and shadowing due to complex environmental structures) and uncertainty (e.g., human activities) can induce significant interference and attenuation in the communication process. The complexities associated with the system and its environment make traditional analytical methods inadequate.

In tandem, advances in computer science and technology have brought about significant enhancements in the performance of computer hardware and software. The ubiquity of computer technology across a spectrum of industries has served to expand its application within the realm of wireless systems. Today, the design, development, and implementation of wireless systems are virtually impossible without the support of computers. Strong computational capabilities underpin the necessary technical support for wireless signal simulation tools.

Extensive research on the application of simulation for wireless system analysis underscores the pivotal role of simulation technology. Robust simulation software, such as MATLAB (MathWorks, 2023), is widely utilized in the analysis of wireless systems, considerably advancing the development of wireless systems.

In the work "Principles of Communication Systems Simulation with Wireless Applications" (Tranter, 2004), the authors classify simulation into deterministic and stochastic categories.

- **Deterministic Simulation:** In deterministic simulation, a deterministic algorithm modulates the simulation parameters and the simulation results are consistent under identical conditions. This type of simulation is generally used to save time by circumventing laborious manual computational processes.
- **Stochastic Simulation:** The parameters of stochastic simulation incorporate random variables that are sampled and predicted through Monte Carlo methods and pseudorandom numbers. The stochastic process allows for the attainment of diverse results for repeated simulations under identical conditions. This form of simulation is particularly suitable for modeling uncontrollable variables and random events within realistic scenarios.

In addition, from a temporal perspective, simulation can be categorized into static and dynamic types. **Static simulation** performs simulations of a state of the system at a specific time, with the simulation model remaining constant over time. On the contrary, the **dynamic simulation** focuses on the impact of changes within the system over time on the results. Dynamic simulation can be further differentiated into **discrete simulation** and **continuous simulation**.

In addition, from the point of view of changes over time, the simulation can be divided into static and dynamic simulation. In **static simulation**, simulation is performed for the state of the system at a specific time and the simulation model does not change over time. **Dynamic simulation**, on the other hand, emphasizes the effect of changes in the system over time on the results. And dynamic simulation can be further subdivided into **discrete simulation** and **continuous simulation**.

- **Discrete Simulation:** In discrete simulation, the parameters of the simulation change discretely over time. That is, these parameters maintain constant over specific time intervals. This characteristic renders repeated simulations of the system over periods where the parameters remain constant redundant. Consequently, discrete simulation updates occur only when the system parameters change. These changes in parameters are typically referred to as events, and such simulations are also known as Discrete-Event Simulation (Banks and Carson, 1986) or Next-Event Simulation.
- **Continuous Simulation:** Continuous simulations require the model parameters to vary seamlessly with time. As such, continuous simulation demands a higher update frequency and more computational power compared to discrete simulation. This type of simulation plays a crucial role for the realization of modern digital twin systems.

The counterpart to simulation is real-world experimentation. Compared to physical experiments, simulations offer several advantages, including lower cost, speed, repeatability, and safety, primarily because they are not constrained by physical conditions and do not interact with the physical world. However, in practice, simulation methodologies and real-world experiments are often seen as complementary approaches, each contributing to a holistic understanding of the subject under study.

The employment of simulation techniques allows for the analysis of wireless systems, independent of various constraints such as hardware devices or deployment environments. For example, the feasibility of a wireless system can be effectively anticipated during the design phase by modeling the simulated system using simulation software. In the deployment phase, environmental modeling provides a comprehensive analysis of the interaction between the environment and the system, offering valuable information on potential challenges. Additionally, simulation provides an affordable platform for trial-and-error attempts and efficient system verification conditions for wireless system analysis. Such efficiencies facilitate the design, development, and implementation of wireless systems in a more rational and streamlined manner.

2.3.2 Wireless Signal Propagation

The most important step in the simulation of a wireless system is the simulation of the wireless signal propagation. An accurate understanding of the behavior of the wireless signal propagation helps to construct high-quality simulation models. The primary medium for wireless propagation is the environment in which the transceivers are located. Due to the lack of transmission lines (e.g. fiber optics and copper wires), different communication links cannot be completely isolated, and the receiver receives mixed signals from different sources, resulting in serious interference. In addition, for signals from the same transmitter, the mixing of signals from different directions at the receiving location can also cause significant interference due to multipath effects. These complex characteristics pose a great challenge to the simulation of wireless signals.

2.3.2.1 Electromagnetic Waves

As discussed in Section 2.2.1, the electromagnetic waves can be described by Maxwell's equations Equation 2.1. Plane waves are the simplest form of electromagnetic waves, with $\mathbf{E}$ denoting the electric field and $\mathbf{H}$ denoting the magnetic field, which are orthogonal to each other and to the Poynting vector $\mathbf{S}_{\text{Poynting}}$ (Poynting, 1884) (i.e., the direction of propagation):

$$\mathbf{S}_{\text{Poynting}} = \mathbf{E} \times \mathbf{H}^* \quad (2.11)$$

In the context of plain waves, the electric field in free space is given by:

$$\mathbf{E} = \mathbf{E}_0 \cos(\mathbf{k} \cdot \mathbf{r} - \omega t + \phi_0) \quad (2.12)$$

where $\mathbf{k}$ represents the wave vector pointing to the propagation direction. ω is the angular frequency, ϕ_0 is the constant phase. And the magnetic field is given by:

$$\mathbf{H} = \mathbf{k} \sqrt{\frac{\epsilon}{\mu}} \mathbf{E} \quad (2.13)$$

where ε is the electric permittivity and μ is the magnetic permeability depending on the propagation medium. And $\mathbf{k}$ is the unit vector of propagation direction. More detailed derivation about the process can be found in a earlier study (Wolski, 2011).

As shown in Figure 2.5, in the "5G Physical Layer Principle" book (Zaidi *et al.*, 2018) the authors mention three types of polarization for plane waves: linear polarization, circular polarization, and elliptical polarization.

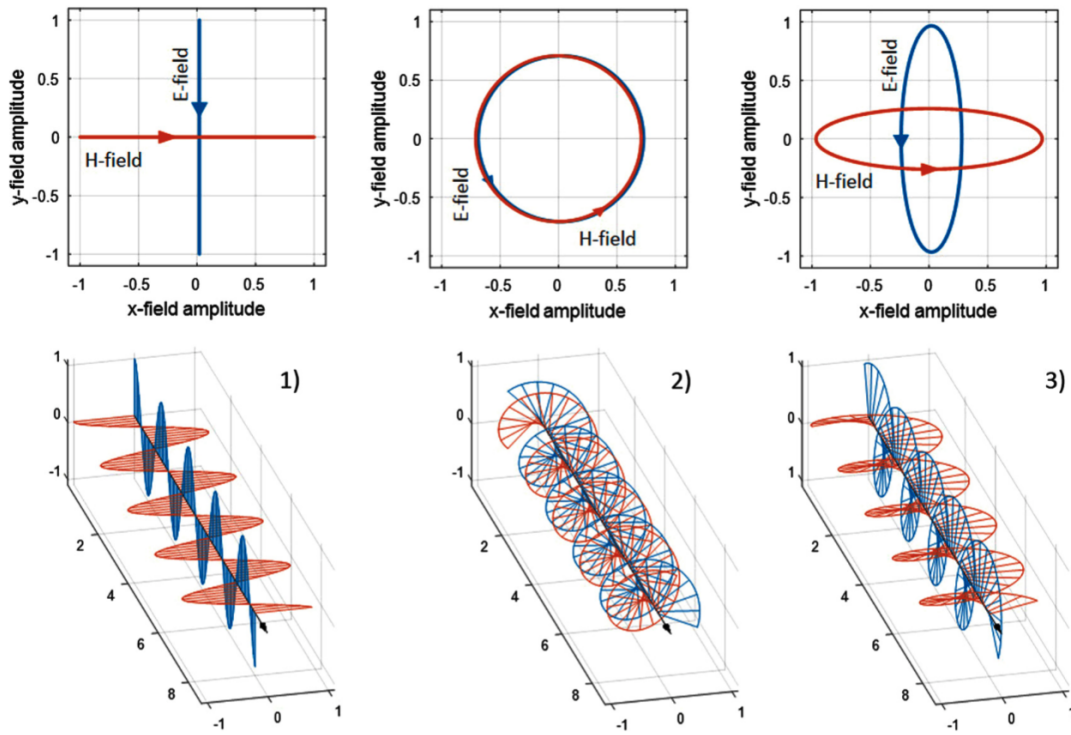


FIGURE 2.5: Plane wave polarizations. Adapted from (Zaidi *et al.*, 2018).

For linear polarization, there is no phase difference between E_x and E_y where $\mathbf{E}_0 = (E_x, E_y, 0)$. For circular polarization, the phase difference is $\pm \frac{\pi}{2}$. For any other cases, where E_x and E_y is different and the relative phase ϕ is nonzero, it becomes elliptical polarization.

2.3.2.2 Propagation Phenomena

Wireless signal propagation focuses on the behavior of signals propagating between transceiver antennas. In general, isotropic antennas are usually assumed for antennas that use isotropic

antennas in order to simplify the study model. Under this assumption, the free-space basic transmission loss can be formulated as:

$$\mathcal{L}_{\text{bf}} = 20 \log \left(\frac{4\pi d}{\lambda} \right) \quad (2.14)$$

where d is the propagation distance, λ is the wavelength of the signal. For arbitrary antenna, transmitter and receiver antenna gains G_t, G_r can be introduced to represent antenna patterns for free-space transmission:

$$\mathcal{L}_f = \mathcal{L}_{\text{bf}} - G_t - G_r \quad (2.15)$$

In reality, the signal propagation environment hardly exists in the true sense of free space; due to the complex spatial structure of the environment and the variety of propagation media, the signal will also be reflected or absorbed by the media. These phenomena can be further subdivided into various types:

- **Absorption** Absorption refers to the absorption of transmitted signal energy by the transmission medium. For example, the signal is absorbed by the atmosphere, or vegetation in an outdoor environment. In indoor environments, there is also a loss of energy caused by the signal traveling through walls.
- **Reflection:** Reflection is the process by which some or all of the components of a signal traveling to the interface of two different media return to the original medium. This generally occurs in planes or near-planes that are large in scale compared to the wavelength (e.g., lakes and walls).
- **Transmission:** Transmission is similar to reflection in that when a wireless signal reaches the surface of a medium, a portion of the component penetrates the medium or is absorbed. Transmission refers to this portion of the component that propagates through the new medium due to refraction. As in an indoor environment, wireless signals can penetrate walls to reach different rooms.

- **Scattering:** Scattering refers to the phenomenon that causes a change in the direction of signal propagation when a signal passes through an inhomogeneous medium. This phenomenon is particularly common in indoor environments due to the characteristics of the building materials.
- **Diffraction:** Diffraction refers to the phenomenon that causes a deviation in the direction of propagation when a signal passes over the edge of a medium, allowing the signal to travel around the back of the medium. For example, a wireless signal can bypass a mountain to reach the area behind it.

This complex signal propagation behavior leads to multipath effects, such that the signal reaching the receiver may be a mixture of signals from the same source through different propagation paths. The difference in propagation paths leads to differences in energy and phase. These differences cause **destructive phase superposition** or **constructive phase superposition** of the mixed signal. This is why, in practice, the signal strength increases with distance at small scales because of constructive phase superposition. This results in a simple path loss that is very different from the true measurement.

2.3.3 Wireless Propagation Modeling

The modeling of signal propagation can be based on statistical or theoretical models (Neskovic *et al.*, 2000). Among them, the statistical model is based on the path loss, while the theoretical model is generally based on the **Finite-Difference Time-Domain (FDTD)** model or the **RT** model.

2.3.3.1 Path Loss Model

The path loss model can be expressed as:

$$\mathcal{L}(d) = \mathcal{L}(d_0) + 10n \log\left(\frac{d}{d_0}\right) + X_\sigma, \forall d \geq d_0 \quad (2.16)$$

here, $\mathcal{L}(d)$ is the path loss at distance d . $\mathcal{L}(d_0)$ is the path loss measured at distance d_0 as reference. n is the path loss exponent dependent on the propagation environment. X_σ is the

representation of random noise that is uncontrollable random events. Path loss modeling is less dependent on environmental surveys and is popular due to its simplicity and computational lightness. However, due to the simplicity of the model, it is not possible to expect a high level of accuracy from the path loss model.

2.3.3.2 Finite-Difference Time-Domain Model

FDTD computes wireless signal propagation by solving Maxwell's equation approximately with finite-difference equations. First, **FDTD** requires meshing of the required computational space. Given the initial conditions, the central difference is used to approximate the derivatives of Maxwell's equation in time and space. Finally, iterative computations are performed at the locations of the grid nodes. The computational requirement of **FDTD** is extremely large, proportional to the size of the computational space, and is generally used for small area simulation tasks.

2.3.3.3 Ray-Tracing Model

RT algorithms compute communication links by modeling the physical phenomenon of signal propagation through space. For complex simulators based on **RT**, scattering as well as complex diffraction phenomena need to be modeled in addition to simple free-space propagation and reflection phenomena. For simulating a signal at a specific spatio-temporal location in space, it is determined by the sum of all the propagation link components coming from it. The use of **RT** algorithms also relies on surveying the propagation environment, where detailed modeling of the environment (e.g. structures, materials, textures) is useful to ensure the accuracy of the simulation results. The computational complexity of the algorithm is also related to the complexity of the environment. With outstanding features such as limitations in small bandwidth and carrier frequency, high spatial resolution, the ability to effectively overcome the limitations of channel measurements and stochastic methods, and the ability to be combined with other modeling methods, **RT** is one of the main methods for the simulation of wireless signals.

2.4 Ray Tracing

RT has seen extensive use in computer graphics for the rendering of digital images of 3D scenes, being highly valued for its superior rendering fidelity. The advent of real-time RT acceleration hardware has further propelled the development and application of RT techniques. RT can simulate a variety of optical phenomena including reflection, refraction, scattering, shadows, and even complex phenomena such as diffraction, dispersion, and caustics. Due to the commonality between wireless and optical signals, both being electromagnetic waves with similar physical properties, RT is also applicable to wireless signals.

The concept of RT can be traced back to the early 16th century, with Albrecht Durer proposing in his book that shadows could be derived using a light source as a viewpoint via perspectival projection (Hofmann, 1990). Later in 1968, Appel (1968) proposed that computers could be used to shade line drawings of solids and calculate shadows utilizing a RT algorithm. Today, RT technology has a broad spectrum of applications, from rendering three-dimensional architectural models in CAD, enabling rendering of real-time scenes in games, to serving as a core technology in wireless signal analysis tools.

2.4.1 The Concept of Ray Tracing

The principle of using rays to model wireless signals stems from an approximation of Maxwell's system of equations at high frequencies (Yun and Iskander, 2015). The use of rays to model wireless signal propagation is based on the following three assumptions:

- In a homogeneous medium, rays propagate in a straight line.
- Rays obey the laws of reflection, refraction, and diffraction.
- Rays carry energy during transmission.

For wireless communication systems, signals are generally considered to extend outward from a point source. This causes the energy carried by the rays to decrease as the propagation

distance increases. This phenomenon is generally described by the inverse-square law. Depending on the signal propagation mechanism, rays are generally classified into the following categories:

- **Direct rays:** Rays traveling from the emitting source directly to the receiver without being obscured. This type of ray requires the existence of a [LoS](#) between the transmitter and the receiver condition.
- **Reflected and scattering rays:** The rays are generated by reflection or scattering phenomena that result in a deviation from the direction of the original incoming rays and return to the medium in which the original rays were emitted.
- **Transmitted rays:** Rays that are generated by entering a new medium and continue in the new medium. These rays are generally associated with reflected rays and obey the law of conservation of energy i.e. the incident energy is equal to the sum of the energy carried by the reflected rays, the energy carried by the transmitted rays, and the energy absorbed by the medium.
- **Diffacted rays:** These rays correspond to the diffraction mechanism. Their modeling is the most complex compared to reflection and transmission. The modeling methods such as [Geometrical Theory of Diffraction \(GTD\)](#) and [Uniform Theory of Diffraction \(UTD\)](#) are widely used to solve diffraction problems.

2.4.2 Ray Tracing Algorithms

[RT](#) algorithms, which predict the propagation path of a wireless signal by tracing the trajectory of ray propagation given the ray's origin, direction, and the structure of the surrounding environment, are grounded in the theoretical principles of Fermat's Principle of Least Time. This principle describes the behavior of rays as they travel between two points within a propagation environment, stipulating that rays invariably follow the path that minimizes the travel time. From this principle, the laws of reflection, refraction, and diffraction can be deduced, which further govern the behavior of [RT](#) algorithms, dictating changes in the propagation direction due to reflection, refraction, and diffraction. Here, four common [RT](#) algorithms are presented.

2.4.2.1 Image Method

Given the positions of the emitter, receiver, and reflecting surface, the position of the reflection point can be obtained by the intersection of the emitter and receiver with the reflecting surface through the intersection of the lines of the mirrored receiver with respect to the reflecting surface. This method is used with multiple reflections of the scene and also because of the recursive structure of the algorithm, making it easy to implement in a computer program. However, in the case of more detailed and complex scene models, the presence of a large number of reflective surfaces makes the computational effort to compute the reflection points grow exponentially, thus limiting its computational speed and the value of its application.

2.4.2.2 Shooting and Bouncing Ray

In 1989, [Ling *et al.* \(1989\)](#) proposed a method based on [RT](#), namely [Shooting and Bouncing Ray \(SBR\)](#), to calculate the interior radar cross section. The core idea of [SBR](#) is to analyze radar cross sections by emitting a large number of rays at the source and tracking the propagation paths of these rays in a partially open cavity. At the time, this was a pioneering approach to applying [RT](#) technology to the field of wireless systems.

[SBR](#) consists of three main components: ray launching, [RT](#), and ray reception. First, all uniformly distributed rays depart from the same emission source and carry the same energy. Next, each emitted ray is individually tracked. The rays can reach the receiver either directly or through interaction with the environment. If the ray is reflected, refracted, or diffracted, a new secondary ray is generated for subsequent tracking. Finally, the mixed signal is calculated by determining whether the ray's corresponding ray tube can reach the final receiver.

2.4.2.3 Ray Casting

Ray casting is the most basic algorithm for image rendering using [RT](#). In computer graphics, the sensor is typically a camera, whose digital form is a grid of closely spaced pixel dots. In the ray-launching phase, ray-casting projects a ray at each pixel for tracing. The color and brightness (i.e. wavelength and energy) of the received pixels are determined based on

information about objects and lights in the scene (i.e. the emitting source). In general, if the ray reaches the light source directly, the pixel corresponds to a color based on the light source and attenuated by distance. If the ray reaches a physical surface and is reflected before it reaches the source, then the color of the pixel is a mixture of the color of the source and the color of the surface, distance attenuation, the law of reflection, and so on. If the ray arrives and cannot reach the light source, then the pixel will correspond to a shadow area in the scene. The biggest advantage of this algorithm is that it can handle complex object structures, but its computational requirements are still relatively large.

2.4.2.4 Path Tracing

Path tracing is a [RT](#) algorithm for image rendering based on the Monte Carlo method. The basic process is similar to the [SBR](#) method, Path tracing integrates the energy reaching the surface of an object as energy from all directions, and then uses the [Bidirectional Reflectance Distribution Function \(BRDF\)](#) method to find its reflection. The Monte Carlo method greatly optimizes the computational load of the [RT](#) algorithm by solving the energy integral by sampling in path tracing. Due to its accuracy and efficiency, it is widely used in 3D image rendering engines. However, in order to obtain high-quality and high frame-rate real-time rendering, the results of path tracing need to be noise-reduced by neural network algorithms. Here is the pseudocode for a basic path tracing algorithm:

CODE 2.1: Basic path tracing.

```
void PathTracing(Ray ray, int depth) {
    if (depth >= MAX_DEPTH) {
        return vec4(0,0,0,0); // skip if ray reach the maximum of tracing depth
    }
    Hit hit = TracingRay(ray);
    if (hit.miss()) {
        return vec4(0,0,0,0); // skip if ray cannot hit any object
    }
    Material material = hit->material;
    vec4 emittance = material.emittance
    // Sample a secondary ray
    Ray nextRay;
    nextRay.origin = hit.location
```

```

nextRay.direction = SampleInHemisphereOf(hit.normal);
float p = 1 / (2 * PI)
// Compute BRDF
float cos_theta = DotProduct(nextRay.direction, hit.normal);
vec4 BRDF = material.reflectance / PI;
// Recursively trace nextRay
vec4 nextColor = PathTracing(nextRay, depth + 1);
// Sum of all the components
return emittance + BRDF * nextColor * cos_theta / p;
}

void PathTracingRenderer(vec4 pixel, int numSample) {
    foreach (i in numSample) {
        Ray ray = GenerateRay(pixel); // generate tracing ray
        color = TracingRay(ray, 0);
    }
    pixel.color /= numSamples; // average over samples.
}

```

2.4.3 Ray Tracing Applications

RT, a sophisticated rendering technique, crafts digital images by meticulously simulating how light interacts with objects. By tracing light's path through a scene, it achieves unparalleled realism in rendering effects. Since its inception, RT has undergone continuous optimization, finding applications in various domains.

- Computer Graphics:** RT has carved a significant niche in the realm of computer graphics, finding its applications in both non-real-time domains such as film and television production, as well as real-time case like video games (NVIDIA, 2023b). At its core, RT is a rendering technique that meticulously simulates the intricate behaviors of light, including reflection, refraction, shadow casting, and indirect illumination, to render scenes with striking realism. It traces rays of light from the camera's perspective, passing through each pixel on the projection plane, and delving into the 3D scene. As these rays traverse the scene, they interact with objects, reflecting off surfaces, refracting through transparent materials, or being obstructed to cast shadows. Each interaction influences the final color and intensity of the

light that eventually reaches the camera. In particular, to improve efficiency, light is often traced in reverse, starting from the camera and looking for light sources. As a fundamental element of modern rendering methodologies, RT has been seamlessly integrated into a variety of 3D engines, including UE5 (EpicGames, 2023), Unity (UnityTechnologies, 2023), and NVIDIA Omniverse (NVIDIA, 2022). These engines equip developers with the tools to produce video games, films, animations, and 3D visualizations that captivate with their photorealistic quality.

- Optical Design:** RT finds a significant application in optical design, where it models electromagnetic wave propagation within optical systems (SYNOPSISYS, 2023). Here, a light ray, represented as a line composed of discrete points, denotes the local wavefront position of a light wave. This ray remains perpendicular to the local wavefront, traveling linearly in a homogeneous medium and altering its direction at refraction and reflection junctures. Adjusts its course based on vector grating diffraction variances at diffraction interfaces and follows the material equation in inhomogeneous media. Such modeling aids in analyzing optical system attributes and verifying the efficacy of the design.
- Acoustic Simulation:** RT has been used for acoustic modeling in various studies (Rindel, 2000; Tsingos *et al.*, 2001). For example, Rindel's study (Rindel, 2000) employed RT for room acoustics simulations, improving our understanding of material and structural scattering properties. Leveraging UTD, RT can depict the diffraction behavior of acoustic waves, facilitating efficient calculations of the acoustic effects of the diffraction path (Tsingos *et al.*, 2001).
- Radio Propagation Modeling:** RT is also pivotal in radio propagation modeling (Yun and Iskander, 2015). Given that radio signals, a subset of electromagnetic waves, exhibit propagation behaviors akin to light, numerous studies have ventured to simulate wireless signal propagation using RT. This approach stands out for its speed and precision compared to traditional methods. RT, when combined with terrain data, architectural structures, and specific parameters of the wireless system, provides exceptionally accurate simulations. Its adaptability also allows for seamless integration with other wireless channel simulation algorithms.

CHAPTER 3

Digital Twin Platform for AI-Enabled Wireless Systems

This chapter investigates the construction of a high-performance wireless system analysis platform based on the digital twin framework. Simulation-based analysis tools have been widely used in wireless system design, development, and maintenance. However, high-precision real-time simulation requires not only high-quality model construction but also high-performance computational efficiency to achieve the real-time simulation requirements. Existing simulation tools do not yet have the capability to perform high-precision large-scale real-time simulation. In this chapter, a high-performance wireless system analysis platform, WirelessDT, is proposed based on the digital twin framework. The principal, design, implementation, and evaluation results of the platform will be discussed in detail. The content of this chapter has been published in the 2022 SIGCOMM Poster and Demo Sessions (Lai *et al.*, 2022).

3.1 Introduction

Traditional wireless signal analysis can be performed through field surveys or computer simulations. Field measurements necessitate surveying a deployed wireless system on-site and evaluating it by situating receivers in various locations to gather and analyze signal measurements. These measurements provide the most accurate representation of the performance of a real system in realistic scenarios. However, this approach is highly labor-intensive and time-consuming and is constrained by preexisting physical conditions such as site and equipment availability, significantly limiting the scope of field surveys.

In contrast, computer simulation offers a greater degree of flexibility and convenience. Simulation tools provide built-in wireless component models and enable the construction of virtual architectural and geographic models. These tools can also intuitively visualize analysis results through simulations within the virtual scene. Although simulation results are generated on the basis of mathematical models and ideal conditions, this can lead to discrepancies with field surveys. However, these discrepancies are typically acceptable. Furthermore, the quality of the simulation is strongly dependent on the accuracy of the modeling, but achieving highly accurate modeling often compromises the simulation speed. This can significantly hinder the quality of real-time simulation. Although some wireless simulation tools can simulate environmental dynamics to some extent, the balance of accuracy and speed often results in poor modeling accuracy. Although this may be acceptable for simulations on large spatial scales, it becomes particularly significant in small-scale indoor scenarios. In addition, due to limitations of the underlying frameworks used, simulation engines are often unable to model complex structures and dynamics, such as human or animal behavior, with high precision, limiting their accuracy in real-time small-scale dynamic simulations.

To address the challenge of fine-grained, high-fidelity dynamic simulation of wireless systems, this paper proposes WirelessDT. WirelessDT is a digital twin platform that provides a simulation environment in real time throughout the lifecycle of wireless systems. Digital twins have recently been used in the industry for design, manufacturing and maintenance (Mihai *et al.*, 2022). In WirelessDT, this study leverages the principles of digital twin to address the aforementioned challenges, following with the creation of a real-time simulation platform. The GPU RTRT (EpicGames, 2022) technique is incorporated for the first time into the digital twin system as the rendering engine to simulate wireless signal propagation and support interfaces for wireless systems. In detail, UE5 (EpicGames, 2023) is selected as the medium for the design and development of WirelessDT. The use of a game engine as the 3D engine and integrating the framework of digital twins allows WirelessDT to achieve high-precision dynamic modeling of physical scenes and leverages the engine's underlying RT rendering pipeline to simulate wireless signal propagation behavior within the scene at high speed. These features enable wireless simulation tools to overcome existing bottlenecks and expand their utility in real-time applications.

3.2 Related Work

The concept of digital twins originated from a presentation at the University of Michigan in 2002, introducing the idea of creating a virtual counterpart of a physical system (Grieves and Vickers, 2017). Later, the term “Digital Twin” was officially attached to the concept. It emphasizes the mirroring of systems between the physical and virtual realms, with data and information flowing between them throughout the entire lifecycle of a product or system. The digital twin has gained recognition in various fields, including astronautics and aerospace, and has been adopted in NASA’s technology roadmaps (Piascik *et al.*, 2010). It has also been proposed for next-generation fighter aircraft and NASA vehicles, addressing challenges and implementation aspects (Glaessgen and Stargel, 2012). Various digital twin platforms have been developed for different industries.

Today, the usage of digital twin platforms has transcended its initial aerospace applications and has permeated diverse industries, highlighting its adaptability and the broad scope of its potential. Various digital twin applications have been developed for various industries (Javaid and Haleem, 2023; NVIDIA, 2023a). In the automotive industry, BMWGroup (2023) and Mercedes-BenzGroup (2023) harness digital twin technology to construct a virtual factory, integrating all existing tools and teams. This technological advancement helps to improve efficiency and optimize new production strategies, leading to substantial cost savings and improvement in production. In the energy sector, the U.K. Atomic Energy Authority (Chapman and Morris, 2019) has developed a full-scale digital twin of fusion reactors to enhance design and development efficiency. Siemens (2017) leverages digital twin technology to establish a unified digital grid model for the Finnish transmission system operator, Fingrid. This approach has yielded benefits such as reduced development time, superior data quality, standardized interfaces, and integration of grid models with asset management, leading to improved decision-making and operational efficiency. Within the infrastructure domain, Ericsson (2022) employs digital twin platforms to address a wide range of needs in network design, optimization, management and maintenance. Buildmedia (2021) has innovated a city-scale digital twin that consolidates open data from various sources for dynamic, real-time

visualization, thus promoting collaboration between different stakeholders. [HOK \(2022\)](#) utilizes digital twin technology for the Centre Block rehabilitation project, incorporating virtual reality to facilitate exploration and interaction with a 3D representation of the Centre Block. [Vouse \(2021\)](#) implements a digital twin system for real-time monitoring and visualization of the position and movement of aircraft at Changi Airport. In the retail sector, Amazon Robotics ([NVIDIA, 2023a](#)) uses a digital twin system to generate photorealistic synthetic datasets that improve the performance of the neural network in autonomous warehouse robots and shorten development times for quicker iterative testing.

WirelessDT stands in contrast to other digital twin solutions through its specialized focus on the design, development, testing, evaluation, and maintenance of wireless systems. It integrates advanced [ML](#) algorithms with a highly specialized [RTRT](#) pipeline and game engine, specifically tailored for wireless simulation tasks. This unique combination enables WirelessDT to surpass existing simulation platforms in the realm of real-time wireless simulation.

3.3 The Design of WirelessDT

WirelessDT is a comprehensive digital twin platform engineered specifically to deliver a high-fidelity simulation environment for wireless signal systems and provide synthesized data to meet the multifaceted demands of wireless system evaluation. Designed with practical applications at the forefront, WirelessDT considers simulation performance, execution efficiency, scalability, and practicability. In alignment with this objective, the digital twin architecture has been incorporated into the system design. This feature empowers WirelessDT to seamlessly traverse the boundary between the virtual and real worlds, interacting with the actual physical system in real-time via an efficient synchronization mechanism. This design philosophy not only enables WirelessDT to integrate more effectively with pre-existing infrastructures but also facilitates a continuous learning process for WirelessDT. This is made possible through real-time interaction with the physical system and perception of the real world, allowing WirelessDT to adapt and optimize its parameters to achieve increased simulation realism. As shown in [Figure 3.1](#), with the principle of digital twin design as a guide, the architecture of

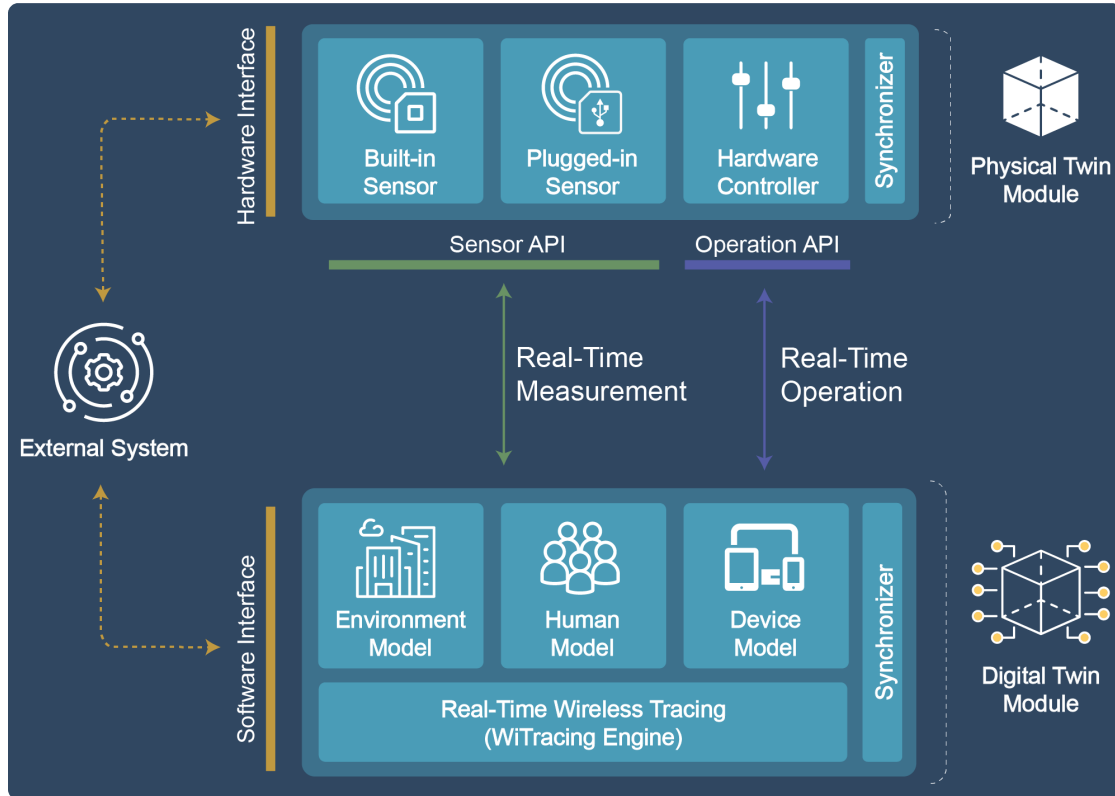


FIGURE 3.1: Architecture of WirelessDT.

WirelessDT is segregated into three modules: the **physical twin** responsible for engaging with the real world, the **digital twin** that hosts virtual representations, and the **communication interface** serving as a bridge between virtual and real realms:

- **Physical Twin** component represents the physical system that will be simulated. It incorporates various sensors to measure the physical state of the system and the surrounding environment. Indeed, while the physical environment itself is part of the physical twin, here the physical twin refers specifically to the wireless system component;
- **Digital Twin** is the virtual representation of the physical system including hardware facilities, physical environment, and human models. It is constructed using a powerful physics engine to accurately replicate the physical environment, system and phenomena;

- **Communication Interface** enables seamless communication between the real and virtual realms, ensuring low latency and high throughput for real-time synchronisation. In addition, the communication interface also provides a convenient way to share data, which allows WirelessDT to work seamlessly with external frameworks or systems.

3.3.1 Physical Twin

The physical twin as the hardware part of WirelessDT is responsible for sensing the real world and providing a physical interface to external systems. The Physical Twin module acquires signals from both internal and external sensors. These signals include measurements of the physical twin's own state such as 3D coordinates, orientation, acceleration, etc., as well as measurements of the environment such as temperature, sound, light (camera sensors), magnetic fields, and wireless signals. By detecting its own state in real time and synchronizing the state data with the digital twin, changes in the state of the physical twin can be reflected in the virtual twin in real time. In this way, it is possible to visualize in the virtual world how the system works in the physical world. On top of this, for physical twins connected to controllable systems, the physical system can be seamlessly controlled by interacting with the virtual twin. This approach facilitates efficient and controllable supervision and management of high-standard unmanned production environments (e.g., automated storage and retrieval system), which is of great significance for future unmanned industries. To this end, a physical twin module consisting of three components: a built-in sensor, a plugged-in sensor, and a synchronization module has been designed as shown in Fig. 3.2.

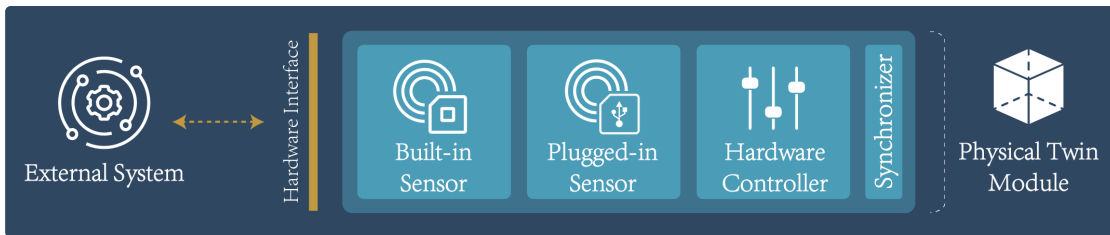


FIGURE 3.2: Architecture of physical twin module in WirelessDT.

3.3.1.1 Built-in Sensors

The built-in sensors provide highly accurate measurements to monitor the state of the physical twin and serve as a reference for the calibration and evaluation of the system. For example, in order to assign the correct position and orientation to a digital entity in a virtual world, the exact state of its physical counterpart need to be known. In this case, sensors for real-time state detection of the physical counterparts are necessary. Another example is when calibrating a WirelessDT system using real physical measurements, where both the position information and the signal measurement information of the physical twin need to be obtained. In this case, high-precision measurements of the physical entity are beneficial to ensure the quality of the system calibration and provide the necessary technical support for subsequent applications.

3.3.1.2 Plugged-in Sensors

The plugged-in sensors from the target system are connected to the WirelessDT system through a physical twin interface. The data captured from the plugged-in sensors are collected and transmitted to the digital twin system to accurately reflect the true sensor state of the target system in real time. In the positioning system, wireless sensors capture wireless signals (e.g. [BLE](#), Wi-Fi, [UWB](#), etc.) and interpret physical parameters (e.g. [RSS](#), [CSI](#), [AoA](#), [ToA](#)) as raw measurements. In practice, by connecting the wireless sensors of the target system to the physical twin, WirelessDT can perform a targeted calibration based on the specific hardware signals and performance. By comparing the raw measurements of the physical sensors with the real-time simulation results, WirelessDT can automatically adjust the digital twin module parameters to optimize simulation fidelity. This targeted and automated calibration feature makes the WirelessDT more compatible with hardware devices from a wide range of brands and vendors, increasing the practicability of the WirelessDT.

3.3.1.3 Hardware Controller

The hardware controller facilitates control of the physical twin or external hardware (e.g., drive units, motors, switches) linked to the physical twin. This control unit is a bidirectional module capable of transmitting real-time operational commands from the digital twin to

the physical twin, subsequently directing these to the drive unit for controlling real-world activities from the virtual world. On the contrary, the hardware controller can relay operator instructions to the hardware through a synchronizer to the digital twin, thereby enabling instantaneous responses and recording of the physical operation processes in the virtual world.

This bidirectional control design fosters a more intimate integration of the virtual and the real. Within the digital twin, control operations are rendered in a comprehensive manner that is not limited to physical displays (e.g., physical indicators and dashboards). The data-rich format of the virtual environment enables deeper operational insights to be illustrated, such as the projected impacts of operations on the system. Meanwhile, enabled by real-time synchronization and operation data logging, the digital twin can trace back historical operations, analyze them in relation to the outcomes of these operations, continuously refine itself and propose superior operational strategies.

3.3.1.4 Synchronizer

The synchronizer module mirrors the real-time operation and sensor measurements for system simulation. The operation and sensor data are synchronized in a bidirectional way depending on the usage.

- During **calibration**, the operation data of the physical twin, including alterations in the position and orientation of the mobile device, are synchronized with the digital twin along with the measurements of the wireless sensor. The digital twin platform generates simulated sensor measurements by emulating the physical environment and phenomena (e.g., wireless signal propagation) with synchronized operation data. When comparing the discrepancies between the real and simulated sensor measurements, the digital twin platform can autonomously modify the system parameters for calibration. In this case, the data flow originates from the real system and moves toward the simulation platform.
- During **evaluation**, the simulated sensor measurements and ground truth position are transferred to the positioning engine deployed on the target device. The positioning engine can execute real-time localization based on simulation data and assess

performance by contrasting the estimation result with the ground truth position. In this scenario, the data flow moves from the simulation platform to the real system.

In conclusion, the physical twin module plays a crucial role in the WirelessDT platform by integrating the wireless components of systems, measuring the state of wireless signals, determining the device's position in the actual environment, and synchronizing this physical state with its digital twin. In practical implementation, a small-scale production system and environment, along with its corresponding digital twin, can be established to validate and calibrate the wireless signal emulation data generated by WirelessDT. The physical twin functions as a versatile sensor, providing insight into the real world. By observing small-scale reality, WirelessDT can adjust its parameters to accurately reflect the physical world and enable precise large-scale simulations.

3.3.2 Digital Twin

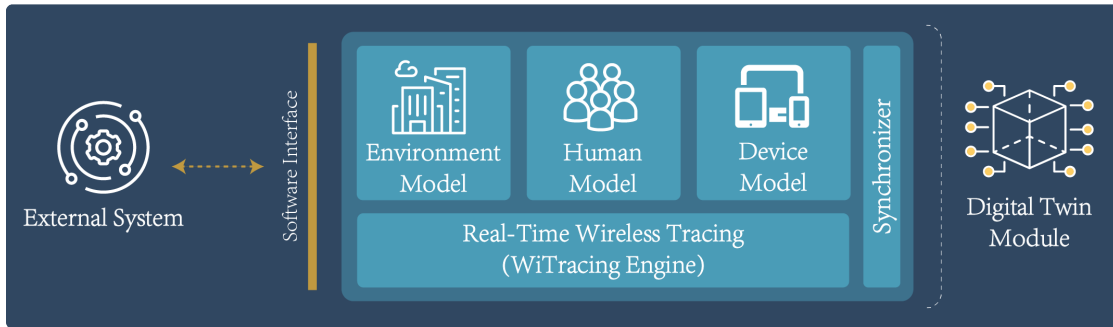


FIGURE 3.3: Architecture of digital twin module in WirelessDT.

WirelessDT's digital twin module is a virtual representation of the physical system and its surroundings. By building virtual counterparts, WirelessDT can better represent the sensor and operational data of the system, the analysis processes, and the analysis results in a more intuitive way. At the same time, the virtual counterparts can operate independently from the real system, which gives the digital twin the freedom of time dimension, i.e., it can be analyzed by backtracking the historical data, and it can be predicted the future state of the system by pre-computing. In addition, the digital twin exists in a virtual space, which allows it to be replicated and executed in parallel, a feature that provides a secure and efficient sandbox

environment for system testing. In WirelessDT, the components of the digital twin can be divided into four different parts, as shown in Figure 3.3.

3.3.2.1 Environmental Model

WirelessDT models physical environments using the advanced 3D game engine [UE5](#), which provides a powerful set of features for building dynamic and realistic scenes using the powerful [GPU RTRT](#) technology. [UE5](#)'s powerful modeling capabilities (geometry, materials, textures) allow us to reproduce highly realistic scenes in WirelessDT. [UE5](#)'s efficient rendering system allows us to simulate and interact in real-time with dynamic and realistic scenes. More importantly, [UE5](#)'s scalability allows us to build digital twins for large-scale scenes.

3.3.2.2 Human Model

In order to introduce environmental dynamics induced by human activities in the digital twin, the human model is included in WirelessDT. These human models can be controlled by [AI](#) as [NPCs](#) that interact with themselves in the scene to simulate human activities (e.g., talking, walking, driving, meeting). They can also be designed as models controlled by humans or external algorithms. This simulation capability allows us to study the impact of human behavior on wireless systems in a highly controlled environment in WirelessDT.

3.3.2.3 Device Model

The device models mimic the physical sensors and the device of reality and provide flexible configurations for simulation. For example, the parameters of the wireless transmitter and receiver model (e.g., transmit power, wavelength, system gain) reflect the configuration of real devices. These sensor settings enable fine-tuning and calibration for simulated devices, which can influence the wireless signal propagation simulation and increase the emulation quality.

3.3.2.4 Real-Time WiTracing Engine

The WiTracing Engine is based on [GPU RTRT](#) and is designed for real-time wireless propagation emulation. To implement WiTracing, a novel rendering pipeline has been innovated to support real-time wireless propagation emulation for wireless propagation modeling based on [UE5](#). A comprehensive solution for the [RT](#) functions, including transmission, scattering, and reflection, is provided. The new pipeline and functions are fully implemented for wireless signal propagation, considering factors such as phase, wavelength, antenna gain, and transmit power. Using the highly optimized [UE5 GPU RTRT](#) engine, WiTracing effectively utilizes [GPU](#) resources to emulate real-time wireless propagation. Moreover, [UE5](#)'s rendering engine enables our platform to visualize "invisible" phenomena (e.g., wireless signal strength distribution) during runtime, which proves valuable for performance analysis.

3.3.2.5 Synchronizer

In contrast to the synchronizer of the physical twin, the synchronizer of the digital twin collects measurement and operational data from the physical sensors and feeds the simulation results back to the physical twin. As the core component of virtual-reality communication, the efficient and reliable communication mechanism of the synchronizer is an important basis for real-time synchronization at high frame rates in WirelessDT.

The collaboration between the physical and digital twins allows WirelessDT to accurately emulate the physical behavior of reality. Leveraging advanced game engine and [GPU RTRT](#) technologies, WirelessDT achieves a high level of precision in replicating the physical behavior of reality at a speed suitable for real-time applications. This remarkable capability unlocks new possibilities for the product lifecycle of wireless systems, facilitating experimentation and iteration within a fully controllable environment, thus aiding in the design, development, and maintenance of wireless systems.

3.3.3 Communication Interface

WirelessDT integrates hardware and software interfaces to facilitate seamless data transfer between external wireless systems. As a platform that serves digital twins, WirelessDT provides convenient [Application Programming Interfaces \(APIs\)](#) for system expansion and integration. For standard wireless systems, integration with WirelessDT [APIs](#) allows existing hardware and software facilities to access WirelessDT and become part of the digital twin. Quick modeling with WirelessDT's in-built wireless prototype model makes it very easy to transform existing facilities into digital twins. In addition, as a high-performance digital twin platform, WirelessDT can generate high-fidelity wireless simulation data in real time, through the provision of [APIs](#) access to WirelessDT can be accessed in real time to obtain a large amount of measurement data, which greatly solves the problem of lack of data for the training and testing of [AI](#) systems that rely on deep learning models and machine models. Structurally, the WirelessDT [APIs](#) can be divided into hardware and software interfaces.

3.3.3.1 Hardware Interface

WirelessDT's hardware interface allows for the integration of external sensors into the physical twin, ensuring compatibility and interoperability with a wide range of sensing devices. Highly accurate simulations cannot be achieved without calibration using real-world observation data. The physical twin's hardware interface allows the WirelessDT to be calibrated and fine-tuned by accessing different wireless devices, ensuring that the system simulation results are consistent with real-world measurements. Furthermore, the hardware interface provided enables [HIL](#) testing, where WirelessDT can control the hardware part for real-time hardware testing.

3.3.3.2 Software Interface

WirelessDT's software interface provides real-time data exchange [APIs](#) that allow wireless systems to be integrated, tested, and evaluated at the network layer. This network layer establishes communication links between the digital twin, the physical twin, and the wireless

system being evaluated, ensuring efficient, low-latency data transfer. Reliable software interfaces provide the basis for communication between internal and external systems. Access to external systems allows WirelessDT to perform [SIL](#) testing of external systems, e.g., training and evaluation of [AI](#) systems. This integrated approach to testing helps simulate the performance of the test system in real scenarios, regardless of environmental constraints. At the same time, the flexibility of virtual space provides low-cost, high-quality, and variable test conditions for a more comprehensive evaluation of system performance.

By incorporating these external [APIs](#), WirelessDT improves its extensibility and usability, allowing it to be easily applied to diverse practical scenarios for wireless systems. This feature enables the seamless integration and evaluation of wireless systems within different environments and systems.

3.4 Proof-of-Concept Implementation of WirelessDT

3.4.1 Implementing the Physical Twin

To construct the physical twin module of WirelessDT, a Raspberry Pi ([RaspberryPi, 2023](#)) is used as the main board to host the physical twin program and connect sensor components. Due to its lightweight, inexpensive, low-power, and scalable properties and a rich set of built-in communication interfaces, the Raspberry Pi emerges as an ideal prototype.

In WirelessDT, a socket server has been implemented as a core service (as illustrated in [Figure 3.4](#)) and deployed it on the physical twin modules. This service integrates sensor component drivers and a software interface to the digital twin. It is responsible for preprocessing raw data, including operations such as filtering and encapsulation. Furthermore, this service ensures data synchronization amidst inconsistent update rates from different sensors. Simultaneously, this service provides an interface for executing operational data of the digital twin, thereby facilitating the users' operations on the digital twin side to be mirrored on the physical twin.

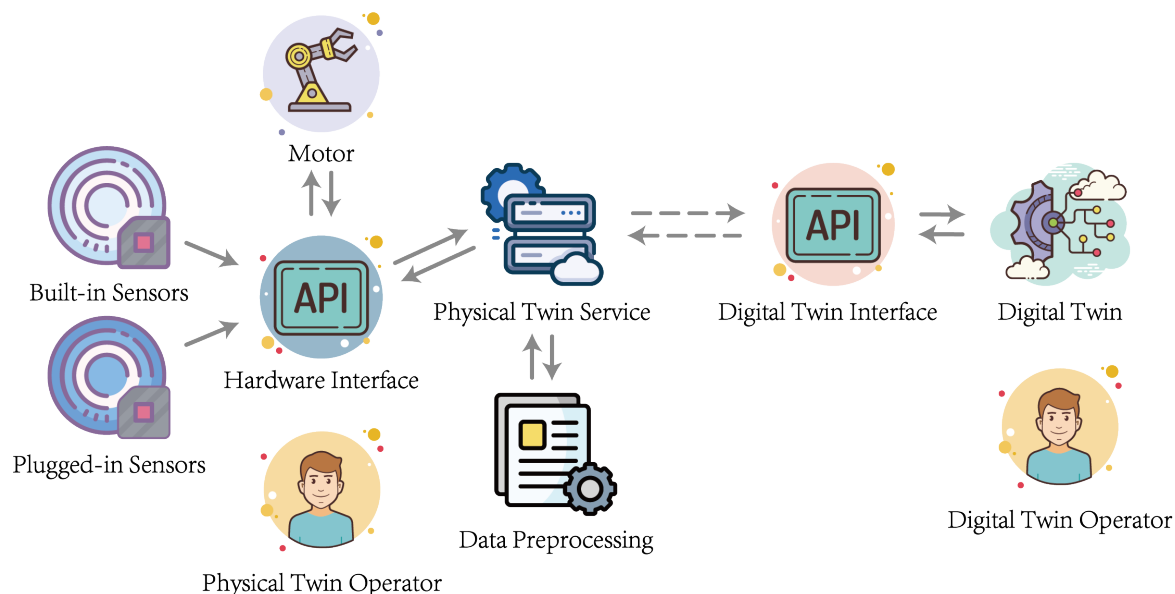


FIGURE 3.4: Physical twin service.

The specific physical hardware used to build the prototype is described in Table 3.1. The wireless sensor, Sseed Studio XIAO nRF52840, functions as the plugged-in sensor in the digital twin, offering [BLE 5.0](#), [Near Field Communication \(NFC\)](#) and [ZigBee](#) measurements. In subsequent chapters, this sensor will be used as a Bluetooth sensor to gather signals for evaluation experiments. The T265, which acts as the built-in sensor, provides high-precision location and [Attitude and Heading Reference System \(AHRS\)](#) measurements to determine the state of the physical twin module. The entire module is powered by a 5V battery, enabling human operators to carry it. Data communication with the digital twin was established through a [WLAN](#) connection. During execution, the physical twin service collects and processes the acquired sensor and operational data and synchronizes it with the digital twin. Concurrently, the operational data from the digital twin is synchronized with the physical twin. The final assembly of the physical twin module is shown in Figure 3.5.

TABLE 3.1: Physical twin hardware components.

Wireless Sensor	Sseed Studio XIAO nRF52840
Location Sensor	Intel® RealSense™ Tracking Camera T265
Mainboard	Raspberry Pi 4 Model B
Power Supply	5V/3A Power bank

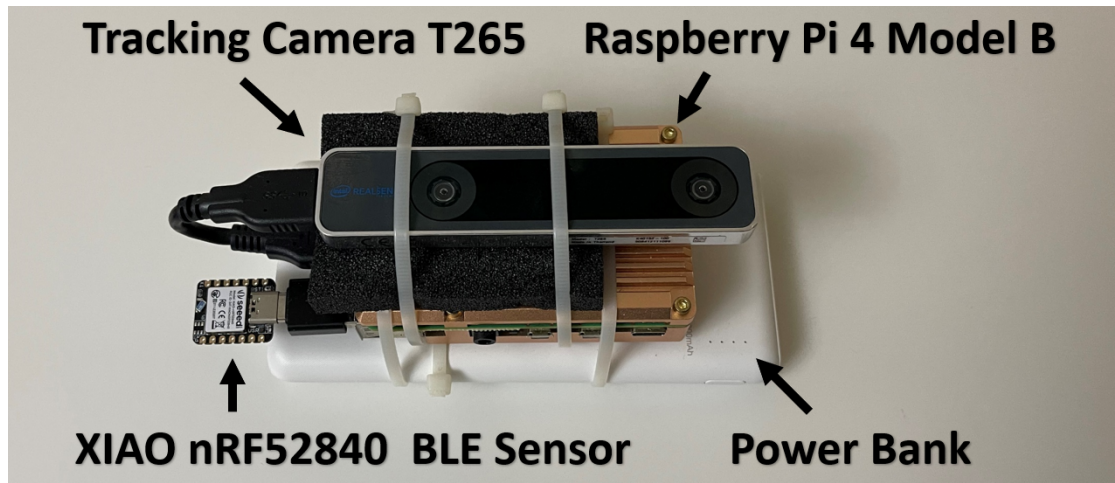


FIGURE 3.5: The proof-of-concept implementation of physical twin.

3.4.2 Implementing the Digital Twin

The primary function of a digital twin is to generate virtual representations of physical systems, which encompasses both the modeling of physical components and the simulation of physical phenomena. This subsection sheds light on the implementation of dynamic environmental modeling and data communication modules, while an in-depth discussion of the core WiTracing engine will be presented in a subsequent subsection. The overview of the digital twin module is shown in Figure 3.6.

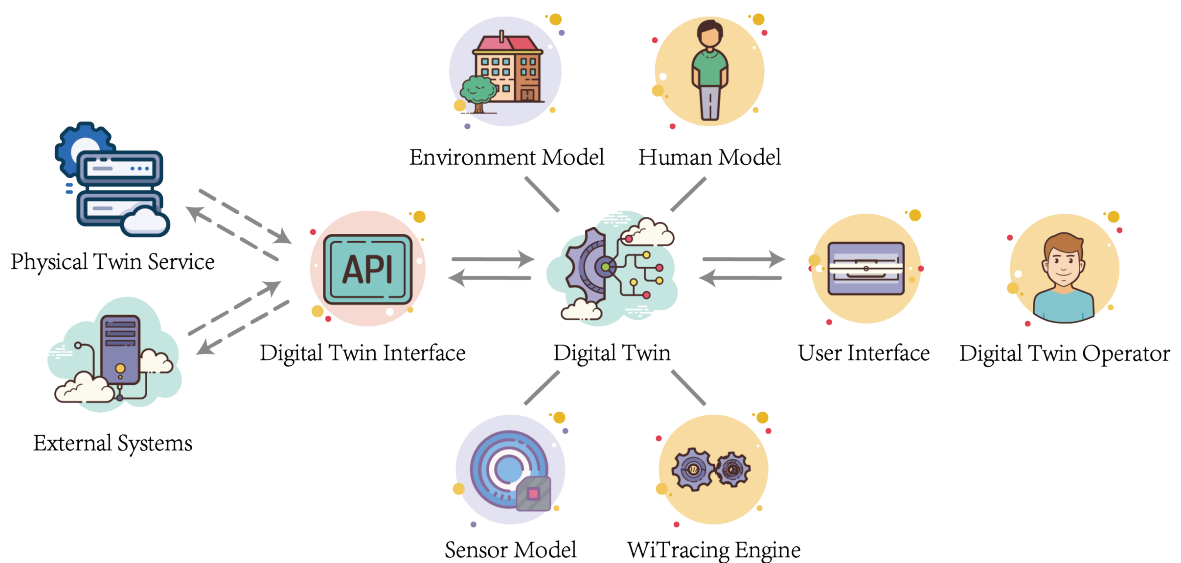


FIGURE 3.6: Digital twin components.

3.4.2.1 Constructing the 3D World

To accurately construct the digital twin, a multitude of information sources were used, including Google Maps, architectural designs (e.g. CAD files), site measurements, and photographs. First, the modeling tools of [UE5](#) were used to produce a 1:1 replica of the physical environment, where a voxel unit in [UE5](#) represents a cubic centimeter in the real world. The constructed building was then fine-tuned according to field measurements. Detailed features such as building walls, doors, and windows were meticulously replicated using photos on site, and models were equipped with corresponding materials (e.g., tiles, paint, blankets) and textures. Upon completion of the structure, furniture and facilities were inserted into the scene to closely mimic the actual environment.

3.4.2.2 Modelling User Behavior

To encapsulate environmental dynamics, [NPCs](#) was incorporated into existing building models. By assigning these [NPCs](#) with animations and [AI](#) logic, human activities such as walking, talking, meeting and working can be effectively simulated. This feature allows for the simulation of the effects of human activities on wireless signals in real-world scenarios. In the realm of wireless propagation simulation, the ability to mimic the dynamic propagation conditions arising from irregular human activities is fundamental to understanding the impact of human behavior on wireless systems.

3.4.2.3 Creating a Data Sharing Interface

To address varying communication requirements, a dual-pronged data interface comprising a socket-based communication interface and a traditional [User Datagram Protocol \(UDP\)](#) communication module has been designed. Accordingly, data synchronization services is established for different modules (e.g., physical twins, mobile devices, [ML](#)). In the case of communication between the mobile device and the physical twin synchronization module, a [UDP](#) server was designed on the digital twin side, which can communicate with the module by establishing a [UDP](#) connection. For the [ML](#) module and other data sharing modules, a

socket server was created, allowing third-party applications to exchange data with the digital twin by establishing a socket connection.

3.5 Summary

This chapter discusses the WirelessDT platform in depth from a conceptual, design, and implementation perspective. This study follows the design concept of digital twins by modeling the virtual counterpart of a physical system, designing and implementing a high-performance real-time wireless simulation engine, and establishing an efficient virtual-reality synchronization mechanism to simulate the wireless system and provide high-fidelity data to serve the real-time applications of the wireless system (e.g., design, implementation, training, testing, evaluation, etc.).

WirelessDT, like digital twins in other fields, breaks the barrier of virtual reality for wireless systems and redefines the life cycle of wireless systems. System validation and testing can be performed independently of the physical environment during the design phase, and by building a virtual sandbox environment, users can perform intuitive solution validation at low cost, safely and efficiently. In the realization phase, implemented components can be independently tested by accessing WirelessDT, a fully functional, high-fidelity virtual environment that allows users to replace unimplemented physical components with virtual components and test integration with implemented components. In the training phase, users can build diverse test scenarios in the virtual space to evaluate the system's performance in multiple ways, independently of the limitations of the real environment. In the test and evaluation phase, WirelessDT can provide real-time simulation data streams, and peripheral test equipment can be interconnected with the virtual environment for real-time testing through the WirelessDT data interface. Although there is an inevitable need to test systems in the field, digital twins can greatly accelerate the production design process prior to real-world deployment.

Subsequent research will focus on large-scale wireless systems, including power stations and automation engineering. The emphasis will be on enhancing simulation diversity, efficiency, and interactivity, enabling WirelessDT's application in both research and industry settings.

For diversity, comprehensive modeling and simulation for a variety of wireless devices, such as Wi-Fi, [UWB](#), and cellular networks, will be conducted. Regarding efficiency, deep neural networks will be employed to automate the modeling of the real 3D world, addressing the cost and efficiency challenges of traditional manual modeling. In terms of interactive performance, a reliable and efficient digital twin system will be developed, based on real large-scale wireless systems. This system aims to provide a convenient and intuitive virtual-reality interaction experience for real systems.

CHAPTER 4

GPU Real-Time WiTracing

This chapter delve into the WiTracing Engine of WirelessDT, with a particular emphasis on WirelessDT's data learning capacities. Despite the existing state-of-the-art wireless analysis tools possessing both wireless simulation and environment modeling capabilities, they fall short in conducting real-time simulations in high-fidelity virtual environments due to technological constraints. To address the problem of dynamic simulation in realistic virtual environments, this chapter describes the design and implementation of our proposed wireless tracing engine, WiTracing, which is based on [RTRT](#) rendering on [GPU](#), as well as the ability to train WirelessDTs on real data to improve the quality of data generation. The content of this chapter has been published in the 2023 ICSE: Companion Proceedings ([Lai et al., 2023](#)).

4.1 Introduction

With the advancement of computer technology, simulation has been widely used both in academia and industry. Because of its ability to model complex systems, it makes it possible to study or analyze problems without physical constraints (e.g., constructing physical prototypes), and allows for a degree of experimentation and testing to be carried out in a cost-effective manner. In the field of education, simulation allows the construction of virtual laboratories where students can learn and experiment in a safe and controlled environment. In the field of scientific research, computer simulation can be used to model climate change, predict the spread of viruses, and so on. In the field of engineering, simulation can be used to study the behavior of complex systems and to validate software and hardware designs with rapid solutions. In manufacturing, automakers can use simulation to test vehicle parameters before

actually building physical prototypes, significantly reducing development time and costs. In the medical field, pharmacokinetic and pharmacodynamic simulations can be used to predict how potential drugs will behave in the human body. In the aerospace field, simulators are used to design and test aircrafts, ranging from modeling the aerodynamic performance of an aircraft to the effects on flight performance in various weather conditions.

Several advanced wireless analysis tools have been equipped with wireless signal simulation and environment modeling capabilities. However, the focus of these tools tends to be on fine modeling of the wireless system itself, such as rich hardware parameter settings, transceiver types, band selection, antenna design, accurate mathematical models, and so on, at the expense of the quality of environment modeling. For wireless signals, the external environment, as the main medium of signal propagation, plays an important role in the operation of the system. Accurate modeling of the external environment is important for studying the impact of the environment on the wireless system.

WirelessDT is dedicated to the high-precision simulation of dynamic wireless systems. A game engine is utilized to construct the physical environment and model both the environment and human activities. Furthermore, the WiTracing Engine and a learning-based calibration process are proposed to ensure real-time simulation and accurate data generation. The WiTracing Engine, built upon the game's rendering engine, integrates [RT](#) algorithms for wireless signal simulation into the game rendering process. This integration allows the [RT](#) computation for wireless signal simulation to be efficiently processed by a [GPU](#) with [RTRT](#) capabilities, significantly enhancing execution efficiency beyond existing simulation tools. Additionally, learning algorithms are designed to train WirelessDT using accurate data. In this approach, WirelessDT is treated as a parameterized model and is fine-tuned using a gradient-based learning algorithm. This method elevates WirelessDT beyond the constraints of traditional mathematical modeling, enabling it to learn underlying parameters in the data and produce high-quality simulation results more efficiently.

4.2 Related Work

4.2.1 Real-Time Engine-Based Simulation

Several solutions, such as Ericsson's 5G simulation platform (Ericsson, 2021) and Microsoft's AirSim (Shah *et al.*, 2018), utilize real-time engines for evaluation purposes. Ericsson utilizes NVIDIA Omniverse (NVIDIA, 2023a) to model realistic scenes and combine them with wireless transmission models to evaluate wireless products. On the contrary, WirelessDT is built on UE5 (EpicGames, 2023), allowing high-resolution dynamic scene modeling and real-time visualization. Unlike Ericsson's simulator, which is limited to the NVIDIA RT renderer for the visual scene, WirelessDT provides an end-to-end solution with a customized wireless signal RT engine to accurately replicate wireless signal propagation. AirSim also uses UE5 for dynamic environment modeling, but does not support wireless signal modeling. The closest works to WirelessDT are CARLA (Dosovitskiy *et al.*, 2017) and AURELION (dSpace, 2023), both designed for the development, testing, and validation of autonomous driving. CARLA is an open-source project that supports various sensors but only uses the CPU RT interface for wireless signal simulation, limiting its performance for intensive wireless simulation tasks. AURELION is a commercial product capable of emulating cameras, LiDAR, and RADAR sensors for SIL and HIL testing. Although AURELION demonstrates real-time simulation of RADAR sensors, details of its implementation remain undisclosed. On the contrary, WirelessDT specifically targets wireless systems, using the GPU RTRT engine to handle intense computational loads during real-time wireless signal simulation. It offers flexible physical and digital interfaces for wireless application integration and supports wireless application life cycles, including design, development, testing, and evaluation.

4.2.2 Ray Tracing-Based Wireless System Simulation

RT is a deterministic approach widely used in wireless signal propagation simulation and channel modeling (He *et al.*, 2019). RT encompasses some outstanding characteristics for the wireless signal, such as fewer restrictions by bandwidth and carrier frequency, high

spatial resolution, effectiveness in overcoming the limits of channel measurements and stochastic methods, and feasibility in combining with other modeling approaches, making it a mainstream method for developing wireless signal simulators. However, the computational complexity of **RT** hinders its application in real-time simulation. This study explicitly investigate the industry-leading implementation in **UE5** and anticipate to address the challenge with the **GPU RTRT** engine. To this end, this chapter presents a novel solution of real-time wireless tracing (WiTracing) engine in WirelessDT for wireless signal simulation. The comprehensive evaluation result demonstrates the extended capabilities of WirelessDT for a new application domain. Moreover, WirelessDT allows computationally intensive tasks, such as real-time training data generation, real-time performance evaluation, and **SIL** testing, providing a new possibility to evaluate the wireless system in real time.

4.3 Preliminary

4.3.1 Ray Tracing in Rendering Engine

This chapter will introduce the real-time WiTracing engine for wireless signal simulation, which is based on the existing **RT** rendering technique. To better explain its principles, this subsection will first introduce the **RT** rendering technique in a real production environment and explain its main principles in detail.

In the game engine (e.g., **UE5** and Unity), **RT** is used to implement its powerful lighting system, that is, to render images captured from camera positions by analyzing the propagation of light emitted from a light source through the virtual scene for the purpose of real-time realistic 3D scenes. As an alternative to raster rendering, **RT** has been combined with traditional rendering techniques and deep learning models to provide a more accurate rendering of images while maintaining frame rates. The nature of light propagation is very similar to the propagation of signals in wireless systems, and therefore, in this paper, an efficient simulation of wireless systems is achieved by referring to an existing **RT** rendering pipeline.

RT is made possible by NVIDIA's invention of the **RT** Cores. The **UE5** accesses the underlying **GPU** hardware resources through the DirectX API released by Microsoft, allowing **RT** to be performed efficiently and used in a variety of areas such as gaming, animation, augmented reality, and virtual reality.

4.3.2 Ray Tracing Pipeline

The simplified DirectX **RT** pipeline in **UE5** can be illustrated in Fig. 4.1.

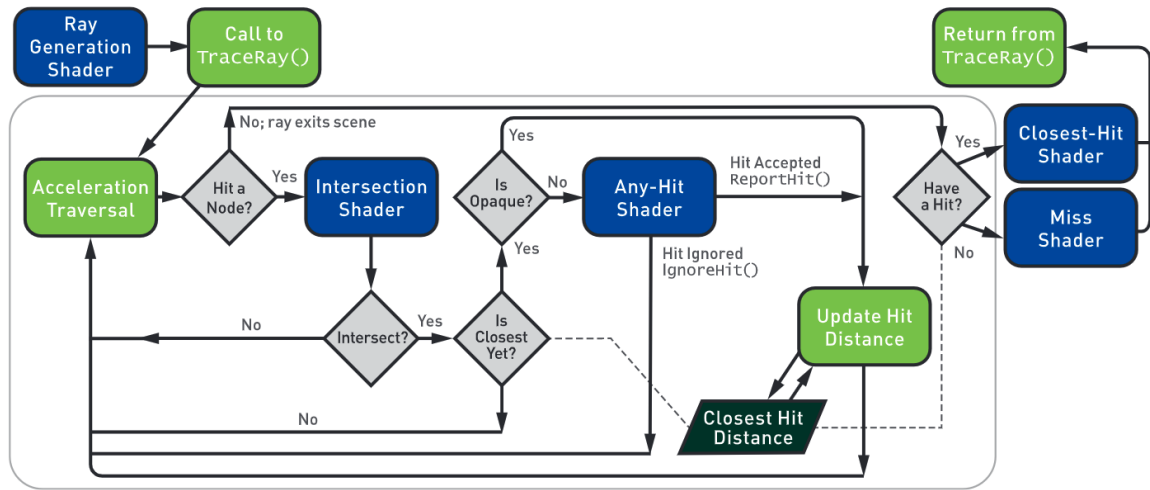


FIGURE 4.1: DirectX ray tracing pipeline. The potential recursive calls are not shown. Adapted from (Haines and Akenine-Möller, 2019).

The **RT** pipeline includes five main shaders: the ray generation, intersection, any-hit, closest-hit, and miss shaders.

- **Ray Generation Shaders** is the starting point of the **RT** algorithm and is responsible for specifying how to generate rays for **RT**. In **UE5**, given the camera information (e.g., position, orientation) and the pixel index, the ray generation shader generates a primary ray based on the current pixel and triggers **RT**.
- **Intersection Shaders** define the computation of intersections of rays with arbitrary primitives. In **RT** based rendering, the performance of intersection computation greatly affects **RT** efficiency. Efficient intersection shaders help to provide reliable underlying support for **RT**.

- **Any-Hit Shaders** are executed at all potential intersections. When the [RT](#) algorithm searches for intersection points, there may be multiple candidates. Any-hit shaders provide an efficient and controllable method of filtering candidate points.
- **Closest-Hit Shaders** are executed on the closest intersection point on each ray and are generally used to calculate the color at the intersection position.
- **Miss Shaders** are executed when the ray does not have an intersection with any primitive in the scene. It is typically used to define the default behavior when there is no intersection for [RT](#).

4.3.3 Rendering by Ray Tracing

With the five shaders, a rendering process using [RT](#) can be illustrated by pseudocode as follows:

Algorithm 1 Ray tracing rendering.

Input: scene $\mathcal{S}$, image $\mathcal{I}$
Output: Rendered image $\mathcal{I}$

```

1: for  $pixel \in \mathcal{I}$  do
2:    $ray \leftarrow \text{GenerateRay}(pixel)$  ▷ Ray Generation Shader
3:    $hit_{closest} \leftarrow null$ 
4:   while  $\mathcal{G} \leftarrow \text{SearchBVHTree}(ray, \mathcal{S})$ 1 do
5:      $hit \leftarrow \text{FindIntersection}(ray, \mathcal{G})$  ▷ Intersection Shader
6:     if  $\text{IsCloser}(hit, hit_{closest})$  and  $\text{IsOpaque}(hit)$  then ▷ Any-Hit Shader
7:        $hit_{closest} \leftarrow hit$ 
8:     end if
9:   end while
10:  if  $h_c$  then
11:     $\mathcal{I}[pixel] = \text{Shade}(ray, hit_{closest})$  ▷ Closet Hit Shader
12:  else
13:     $\mathcal{I}[pixel] = \text{Miss}(ray)$  ▷ Miss Shader
14:  end if
15: end for

```

In [UE5](#), [RT](#) based rendering is implemented via path tracing with much the same algorithmic process. In the subsequent subsections of this chapter, it will be shown how path tracing

¹The [Bounding Volume Hierarchy \(BVH\)](#) is a tree data structure that forms all geometric objects into a tree leaf node. The bottom bounding nodes are connected by larger upper bounding nodes, and this tree structure can effectively improve the query speed of collision detection in [RT](#) computation.

can be used to implement high-performance wireless signal simulation based on rendering techniques.

4.4 Real-Time WiTracing Engine

To ensure high-performance simulation, the GPU RTRT engine is utilized to simulate the physical behavior of wireless signal propagation in 3D dynamic scenes. It is believed that by leveraging RT graphical rendering engines, simulation data can be generated with sufficient speed to support real-time applications such as training, testing, and evaluation. To achieve this objective, the UE5 source code was modified, and a new RT rendering pipeline, WiTracing, was developed, specifically for wireless propagation simulations. This section first outlines the process of generating tracing rays for wireless signal simulation, followed by a detailed explanation of how wireless signal measurements are calculated in the WiTracing process.

4.4.1 Sensor Simulation

UE5 uses pinhole perspective projection to simulate camera sensors to generate tracing rays, which is the most fundamental step in the RT process. However, this pinhole perspective projection is only suitable for sensor with flat sensing area such as camera sensors. In comparison, wireless sensors can capture signals from all directions on the sphere. To simulate such behavior, a spherical projection approach is required. Although spherical Fibonacci mapping (Keinert *et al.*, 2015) can generate uniformly distributed rays, the coordinate information becomes confused after the mapping process, making it difficult to visualize and understand. Therefore, this study choose the octahedral mapping (Meyer *et al.*, 2010) of the sphere for the generation of rays. In contrast to Fibonacci mapping, octahedral mapping can map the sphere into 2D space with relatively low distortion while maintaining the relative position information unchanged. Figure 4.2 shows an octahedral mapping process, while step 2 in Figure 4.3 shows the RT result from a wireless sensor perspective. The implementation of octahedral mapping for generating the primary WiTracing ray in the unreal shader is shown in Code 4.1

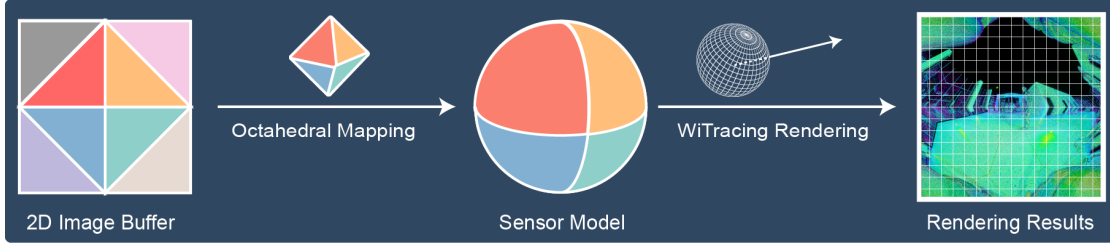


FIGURE 4.2: The octahedral mapping.

CODE 4.1: The octahedral mapping for ray generation.

```

RayDesc GenerateOctahedralRay(float2 Pixel, float2 Dim) {
    float2 UV = 2*(Pixel/Dim)-1;  # normalizes the pixel coordinates
    # converts the 2D UV into a 3D direction vector using octahedral mapping
    float3 Direction = float3(UV.x, UV.y, 1.f - abs(UV.x) - abs(UV.y));
    if (Direction.z < 0.f) {
        float2 Sign;
        Sign.x = (Direction.x >= 0.f)?1.f:-1.f;
        Sign.y = (Direction.y >= 0.f)?1.f:-1.f;
        Direction.xy= (1.f - abs(Direction.yx)) * Sign.xy;
    }
    RayDesc Ray;
    Ray.Direction = normalize(Direction.zxy);
    Ray.Origin = LWHackToFloat(MakeLWCVector3(View.ViewTilePosition, View.
                                                RelativeWorldCameraOrigin));

    return Ray;
}

```

4.4.2 Wireless Propagation Simulation

To simulate wireless propagation in 3D dynamic scenes, this study take advantage of the UE5 built-in GPU RTRT Engine. The RTRT Engine, originally designed for graphics rendering, simulates the propagation behavior of light in virtual environments. Compared to visible light, the propagation of wireless signals exhibits similar physical behaviors such as direct and indirect transmission, reflection, and scattering. The rationale of using RT to simulate the propagation of wireless signals has been proven by numerous investigations (He *et al.*, 2019). However, most existing RT-based wireless signal simulators cannot support real-time simulation with realistic 3D scenes due to the intensive computation required for RTRT.

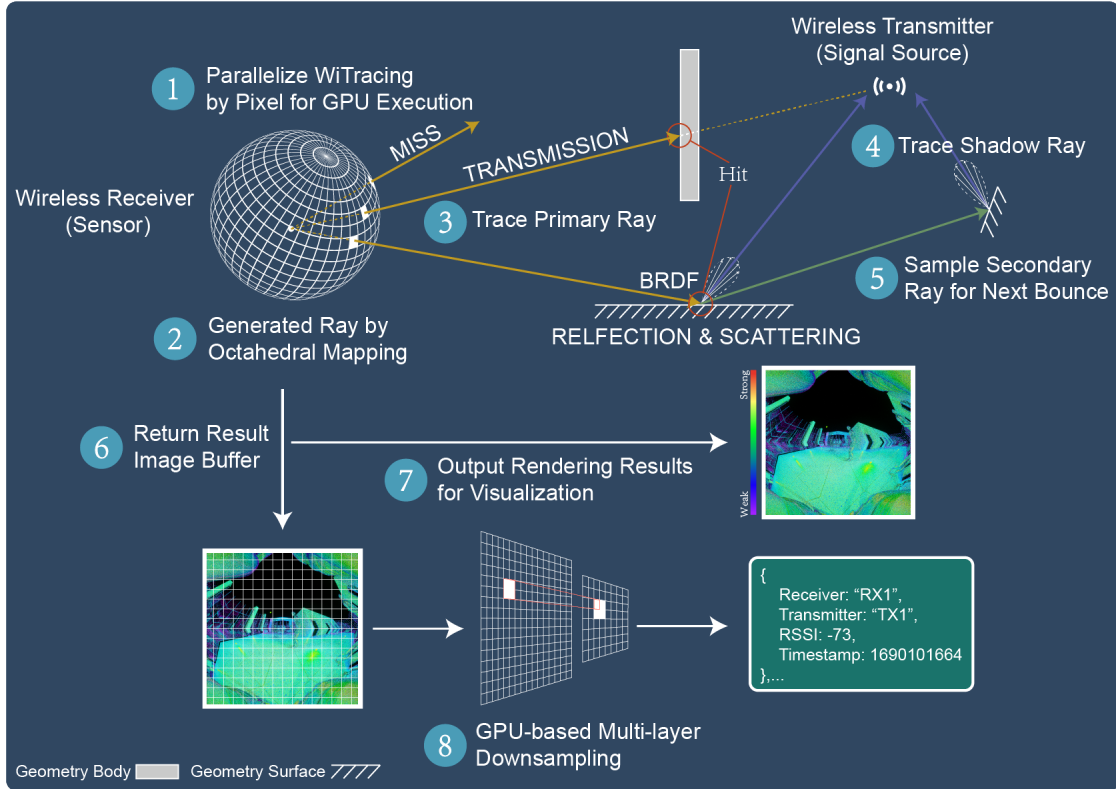


FIGURE 4.3: The process of WiTracing.

In WirelessDT, the WiTracing rendering pipeline derived from UE5's RT pipeline is developed, dedicated to wireless signal propagation simulation. In WiTracing, the wireless **Receiver (Rx)** and **Transmitter (Tx)** correspond to the camera sensor and light source in the original RT pipeline. The primary tracing rays are first shot from the wireless position **Rx** in all directions. Based on the intersection position, material, and sampling results of the ray/surface, the WiTracing process is divided into different situations corresponding to the three major physical behaviors of wireless signal propagation, as shown in Fig. 4.3 and the following:

- **Transmission:** In this case, the wireless signal can propagate to the **Rx** either directly or indirectly (i.e., penetrating through transmission media). The propagation distance of the signal between **Rx** and **Tx** corresponds to the distance between the origin of the primary ray and the position of **Tx**. The loss of signal incurred as a result of transmission through the medium is governed by the penetration rate in our implementation.

- **Scattering and Reflection:** If the ray intersects with any object, the ray will be scattered or reflected according to the material characteristics (e.g., roughness, metallic, texture). The direction and energy reflected or scattered are determined by the BRDF (Nicodemus, 1965) and the Multiple Importance Sampling (MIS) (Veach and Guibas, 1995) algorithms.

$$\underbrace{L_r(\mathbf{P}, \omega_o)}_{\text{Reflected Energy}} = \int_{\Omega} \underbrace{L_i(\mathbf{P}, \omega_i)}_{\text{Incident Energy}} \underbrace{f(\mathbf{P}, \omega_o, \omega_i)}_{\text{BRDF}} \underbrace{(\mathbf{n} \cdot \omega_i)}_{\text{Cosine}} d\omega_i \quad (4.1)$$

The calculation of reflected energy at the intersection point is based on Equation 4.1. $\mathbf{P}$ is the intersection position of the ray/surface. ω_i and ω_o are the incident and reflected directions of the ray. $\mathbf{n}$ is the normal vector at the surface point $\mathbf{P}$, which describes the texture of the material. L_i is the incident energy and f is the BRDF, which describes the reflection behavior at the intersection point. The total reflected energy L_r toward ω_o is the integral over all incident directions that come over the hemisphere Ω around the normal. MIS is then applied to solve the integral.

- **Miss:** If the ray cannot intersect with any surface in a given direction or within the maximum propagation distance, the tracing process is terminated.

If the tracing process is not terminated by certain conditions (reaching the maximum number of bounces, reaching the maximum propagation distance, termination by the Russian roulette random process, or the energy being too weak to be detected), a secondary ray is generated and the tracing process is repeated. Given the positions of all intersection points and Tx-Rx pairs, the distance attenuation can then be calculated using the inverse square law. This study simplify the power equation, Equation 4.2, by assuming that all antennas are isotropic, emitting energy evenly in all directions.

$$P_r = \frac{P_t}{4\pi d^2} \quad (4.2)$$

Subsequently, all RT results originating from the same primary ray and Tx are aggregated. During this step, all RT results for a single Rx are stored and mapped to a 2D image space. Fig. 4.3c shows a WiTracing result for a single Tx-Rx pair. The color of each pixel represents

the energy level in the corresponding direction, where red means a strong signal, and purple indicates a weak signal.

4.4.3 WiTracing Algorithm

Algorithm 2 GPU real-time WiTracing rendering.

Input: Image cache $\mathcal{I}_{\text{RSS}}, \mathcal{I}_{\phi}$ for storing the rss and phase, 3D scene $\mathcal{S}$, wireless transmitter tx and receiver rx , maximum number of bounces b_{max} .

Output: Image cache $\mathcal{I}_{\text{RSS}}$ and $\mathcal{I}_{\phi}$.

```

for  $pixel \in \mathcal{I}_{\text{RSS}}$  do in parallel
   $ray \leftarrow \text{GenerateRaybyOctahedralMapping}(pixel, rx)$  ▷ Ray Generation(Code 4.1)
   $rss, \phi \leftarrow 0, 0$ 
   $b \leftarrow 0$ 
  while  $b \leq b_{\text{max}}$  do
     $hit \leftarrow \text{TraceTransmissionRay}(ray, \mathcal{S})$ 
     $rss, \phi \leftarrow \text{MixSignalFromDirectOrIndirectTransmission}(rss, \phi, hit)$ 
    if  $\text{IsMiss}(hit)$  then ▷ Miss: nothing was hit
      break
    else
       $ray_{\text{shadow}} \leftarrow \text{GenerateShadowRay}(hit, tx)$  ▷ Shadow ray pointing from  $hit$  point to  $tx$ 
       $hit_{\text{shadow}} \leftarrow \text{TraceShadowRay}(ray_{\text{shadow}}, \mathcal{S})$ 
      if  $\text{IsMiss}(hit_{\text{shadow}})$  then ▷ The shadow ray can reach transmitter
         $brdf, mis \leftarrow \text{EvaluateBRDFAndMIS}(ray, ray_{\text{shadow}}, hit)$ 
         $rss, \phi \leftarrow \text{MixSignalFromScatteringOrReflection}(rss, \phi, brdf, mis)$ 
      end if
       $ray.\text{origin} \leftarrow hit.\text{location}$  ▷ Use current hit point as origin for next bounce
       $ray.\text{direction} \leftarrow \text{SampleDirection}(ray, hit)$  ▷ Sample new direction for next bounce
    end if
     $b \leftarrow b + 1$ 
  end while
   $\mathcal{I}_{\text{RSS}}[pixel], \mathcal{I}_{\phi}[pixel] \leftarrow rss, phi$ 
end for

```

The GPU real-time WiTracing rendering process is elucidated in Algorithm 2. Within this process, the generated RSS and phase simulation results are stored in image data structures, namely $\mathcal{I}_{\text{RSS}}$ and $\mathcal{I}_{\phi}$. Each pixel from these structures corresponds to a measurement from a given direction, as defined by the ray generation algorithm. The function **GenerateRaybyOctahedralMapping** creates a tracing ray through Octahedral Mapping, based on the rx position and the current pixel index, during parallel execution.

The procedure **TraceTransmissionRay** traces the ray in the current scene $\mathcal{S}$ to identify potential hit points and the corresponding payload hit . The payload includes information

such as hit point position, material, texture, and more. **MixSignalFromDirectOrIndirectTransmission** aggregates the signal throughout the transmission process, for example, if the ray penetrates the **Tx** body before reaching the hit location, it becomes necessary to account for the contribution and attenuation of the signal due to penetration.

Upon locating a valid hit point, **TraceShadowRay** traces a ray—generated by **GenerateShadowRay**—from the current hit point to **Tx** tx . If the result hit_{shadow} shows that the shadow ray can reach tx without complete obstruction (e.g., if part of the energy can penetrate the geometry to reach tx), then **EvaluateBRDFAndMIS** calculates the **BRDF** and **MIS**. Subsequently, the contribution at the current point is accumulated at rss and ϕ .

Following that, **SampleDirection** generates a new direction through a sampling process based on ray and hit payload to update ray . The origin of ray is reset to the hit position for the next tracing loop until the maximum number of bounces is reached. In this context, this study assume that wireless signals are sinusoidal waves. To amalgamate signals reaching the same position with different amplitudes and phases and return the radiance of the mixture, the shader code is displayed in Code 4.2.

CODE 4.2: The WaveSum code for accumulated sinusoidal signal.

```
void WaveSum(float Amp1, float Pha1, float Amp2, float Pha2, inout float OutAmp, inout
            float OutPha) {

    float x1 = Amp1 * cos(Pha1);
    float y1 = Amp1 * sin(Pha1);
    float x2 = Amp2 * cos(Pha2);
    float y2 = Amp2 * sin(Pha2);
    float x = x1 + x2;
    float y = y1 + y2;
    OutAmp = sqrt(x * x + y * y);
    const float eps = 0.000001f;
    x = x > -eps && x < eps ? eps : x;
    OutPha = atan2(y, x);
}
```

4.4.4 Parallel Data Aggregation

In reality, wireless sensors output a single value for each measurement. As a result, a parallel multilayer downsampling process is designed to aggregate the measurements from all directions, as illustrated in Figure 4.4. The number of downsampling layers is determined by $\lfloor \log_2 x \rfloor$, where x is the side length of the 2D image buffer used to store the WiTracing results from wireless propagation simulations. For each downsampling layer, four adjacent pixels is sampled into one, with subsequent downsampling processes based on the results of previous layers. The final layer outputs a single value that represents the current sensor measurement. Ultimately, the sensor measurement is converted to the dedicated unit as the emulation result. The multilayer downsampling algorithm is shown in Algorithm 3.

Algorithm 3 Multi-layer downsampling.

Input: Image cache $\mathcal{I}_{\text{RSS}}, \mathcal{I}_{\phi}$ for rss and phase from WiTracing results.

Output: rss and ϕ .

if HasSameSize($\mathcal{I}_{\text{RSS}}, \mathcal{I}_{\phi}$) **then**

$d \leftarrow \mathcal{I}_{\text{RSS}}.\text{size}$

▷ The length of side of image buffer

$n \leftarrow \lfloor \log_2 d \rfloor$

▷ Calculate the number of layers

$\mathcal{I}_{\text{RSS-prev}}, \mathcal{I}_{\phi\text{-prev}}, \text{size}_{\text{prev}} \leftarrow \mathcal{I}_{\text{RSS}}, \mathcal{I}_{\phi}, d$

$\mathcal{I}_{\text{RSS-sample}}, \mathcal{I}_{\phi\text{-sample}} \leftarrow \text{null}, \text{null}$

for $i \in n$ **do**

$\text{size} \leftarrow \text{RoundUpToPowerOfTwo} \left(\frac{\text{size}_{\text{prev}}}{n} \right)$

$\mathcal{I}_{\text{RSS-sample}} \leftarrow \text{InitImageBuffer}(\text{size})$

$\mathcal{I}_{\phi\text{-sample}} \leftarrow \text{InitImageBuffer}(\text{size})$

for $(x, y) \in \mathcal{I}_{\text{RSS-sample}}$ **do in parallel**

$\mathcal{I}_{\text{RSS-sample}}[x, y], \mathcal{I}_{\phi\text{-sample}}[x, y] \leftarrow \text{MixSignal}(\mathcal{I}_{\text{RSS-prev}}[2x, 2y], \mathcal{I}_{\phi\text{-prev}}[2x, 2y],$

$\mathcal{I}_{\text{RSS-prev}}[2x + 1, 2y], \mathcal{I}_{\phi\text{-prev}}[2x + 1, 2y],$

$\mathcal{I}_{\text{RSS-prev}}[2x, 2y + 1], \mathcal{I}_{\phi\text{-prev}}[2x, 2y + 1],$

$\mathcal{I}_{\text{RSS-prev}}[2x + 1, 2y + 1], \mathcal{I}_{\phi\text{-prev}}[2x + 1, 2y + 1])$

end for

$\text{size}_{\text{prev}} \leftarrow \text{size}$

$\mathcal{I}_{\text{RSS-prev}}, \mathcal{I}_{\phi\text{-prev}} \leftarrow \mathcal{I}_{\text{RSS-sample}}, \mathcal{I}_{\phi\text{-sample}}$

end for

$\text{rss}, \phi \leftarrow \mathcal{I}_{\text{RSS-sample}}, \mathcal{I}_{\phi\text{-sample}}$

end if

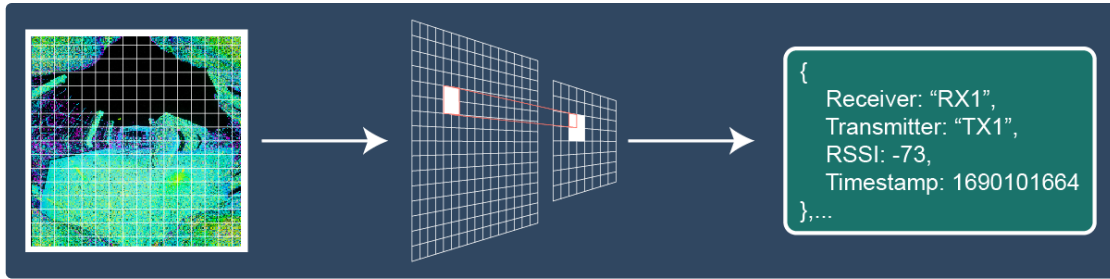


FIGURE 4.4: The multi-layer downsampling for data aggregation.

4.4.5 Data Caching and Visualization

This study use a 2D image (i.e., 2D render target) to cache the intermediate WiTracing data, allowing us to visualize the WiTracing results directly in real time. To enhance the visual effect, the energy values is mapped to different colors, ranging from red to purple, as illustrated in Figure 4.5.

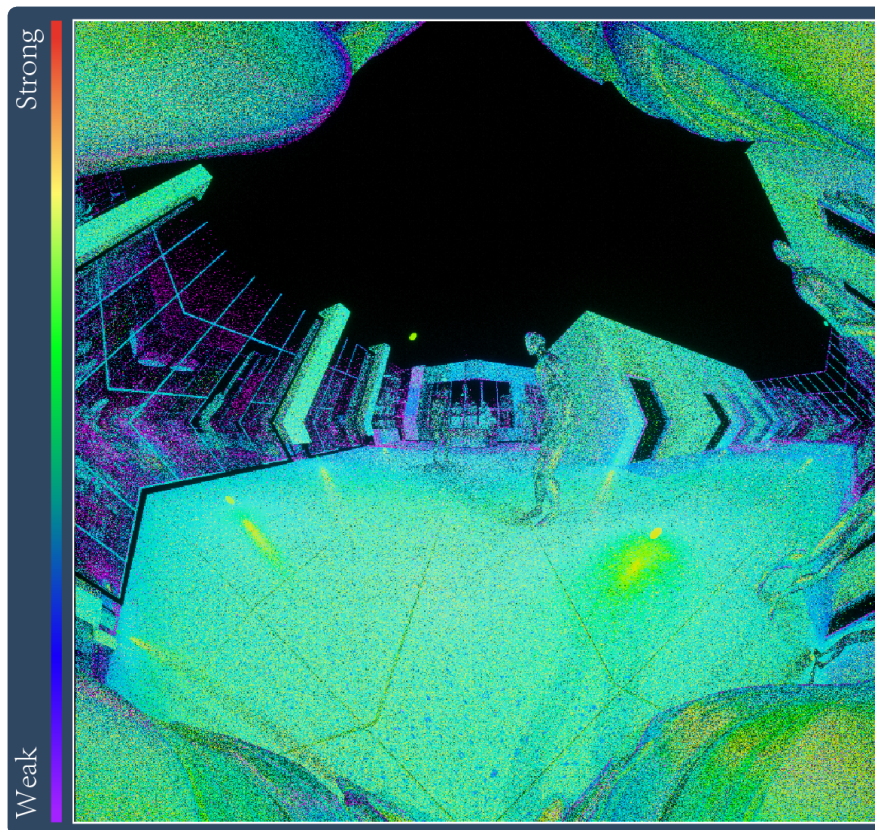


FIGURE 4.5: The visualization of WiTracing results.

4.5 Learning-Based Calibration

WirelessDT's foremost objective is to execute a rigorous and precise simulation of real wireless systems. This task focuses on ensuring the fidelity of the simulation process and the ensuing results. Fidelity, within a simulation context, denotes the accuracy and realism with which the simulation depicts the system or process it is modeling. Broadly, fidelity can be categorized into the following.

- **Physical Fidelity:** This aspect accurately mirrors the physical characteristics of the real world. For example, in WirelessDT, it involves the precise simulation of the wireless signal propagation process.
- **Functional Fidelity:** This facet ensures realistic functional interactions with external systems. When wireless mobile applications are tested using WirelessDT, external mobile devices can interface with the simulation platform through both software and hardware, thereby facilitating interactive real-time tests like [SIL](#) testing. In such scenarios, WirelessDT can effectively substitute the actual test environment.
- **Data Fidelity:** A critical responsibility of WirelessDT in the simulation process is to generate data that closely mimic real data at all times. The quality of these data directly impacts the reliability and utilitarian value of the simulation system.

Typically, simulation systems with higher fidelity demand greater complexity and, consequently, their development and operational costs are elevated. High-fidelity simulations largely depend on an in-depth understanding of physical phenomena and precise mathematical modeling. Nevertheless, in certain fields like astronomy, the limitations in human understanding of physical phenomena hinder the creation of accurate mathematical models. Moreover, the mathematical models deployed in simulations are typically ideal models subject to specific preconditions, resulting in inconsistencies between simulation results and actual observations. The complete modeling of complex systems is impractical, prompting the usage of simplified mathematical models. Adding to these complexities, the absence of modeling for stochastic events, such as human activities, from real physical scenarios further undermines the simulation's accuracy.

To address these challenges, a learning-based calibration framework is proposed. Under this framework, WirelessDT can learn real-world physical processes and their underlying variables from real acquisition data, thus fitting the discrepancy between mathematical modeling and real processes. This ability to learn from the real world allows WirelessDT to further improve the quality of simulation while being more flexible, introducing a strong plasticity to WirelessDT. The principles, design, and implementation of this learning framework are specified in this section.

4.5.1 Calibration with Real World Data

WirelessDT provides two calibration techniques to better align with real-world scenarios. The first method involves utilizing an offline dataset, allowing WirelessDT to learn from the data and estimate optimal configurations for simulations. Alternatively, the second method enables real-time sensor calibration, connecting physical sensors to the WirelessDT platform through a hardware interface, and synchronizing real-time measurements for precise calibration.

4.5.1.1 Calibrating with Offline Dataset

The calibration process aims to enable WirelessDT to learn and reverse engineer latent states, parameters, and characteristics from real data. To adapt the offline dataset for WirelessDT and generate simulation data that closely resemble the original dataset, a learning-based fitting procedure is designed. This method not only allows WirelessDT to generate data that closely mirror the initial dataset, but also creates simulation data for areas or scenarios not covered by the original data, leveraging environmental modeling and the WiTracing Engine. The detailed fitting process can be divided into four steps:

- **Data Preprocessing:** Data preprocessing involves reducing the noise in raw data, filtering out abnormal data points, and improving data quality. For example, when using [RSS](#) for estimation, measurements that are too weak may contain significant

noise, leading to substantial estimation errors. Implementing appropriate mechanisms to filter out unreliable measurements before training improves the quality of estimations.

- **Initializing Estimation:** To initiate training, initial estimations of the states and parameters for the training targets need to be generated. In indoor positioning tasks, for example, determining the position and power of signal sources (such as [BLE](#) beacons or Wi-Fi) using [RSS](#) measurements. In this case, the initial estimation can be achieved by combining least squares methods with the path loss model. The loss is calculated using the observation position, [RSS](#) measurements, and estimated signal source states.
- **Training:** During the training process, the entire digital twin system is incorporated as a training model, introducing the WiTracing Engine into the inference pipeline. Unlike conventional training models, including the digital twin system as part of the training model results in a non-differentiable loss function. To address this issue, the Monte Carlo method is employed with a Gaussian distribution to sample the target states. For each pair of state sample and training data, WirelessDT calculates the loss separately and updates the estimation with the state that yields the minimal loss.
- **Evaluation:** To evaluate training quality, the initial measurement is used as a baseline and compared with the training result. During the evaluation, both the baseline estimate and the training result serve as configurations for WirelessDT. The corresponding simulation data is generated using the baseline and training configurations separately for each data point from the training data. The simulation data is assessed against the raw dataset using multiple metrics, including [Mean Square Error \(MSE\)](#) and [Kullback-Leibler Divergence \(KL Divergence\)](#). A high-fidelity simulation dataset should exhibit a slight absolute difference and a similar distribution to the raw dataset.

In summary, the learning-based calibration techniques in WirelessDT enable the creation of a high-fidelity virtual environment that closely resembles real-world scenarios. By leveraging real-world data and incorporating online and offline calibration methods, WirelessDT

can generate accurate simulations for the evaluation and development of wireless software applications.

4.5.1.2 Calibrating with Online Measurement

WirelessDT also offers online calibration functionality. Its physical twin components provide a hardware interface to connect real-world sensors. During online calibration, these sensors operate within the WirelessDT platform, measuring the physical world. The real-world data are then synchronized with the digital twin components. Similarly, the digital twin generates emulated data based on the built-in sensor readings. For example, in an indoor positioning scenario, the physical twin can measure its 3D coordinates using built-in high-precision sensors, along with signal strength measurements from external [Rxs](#). The digital twin synchronizes the sensor positions based on the 3D coordinates and generates emulation measurements in the virtual world. The emulation results are subsequently compared with real-time physical measurements. Based on this comparison, WirelessDT can optimize its parameters and maximize the fidelity of the generated data. This feature is particularly useful in applications that require real-time calibration.

4.5.2 Formulation

The core function of WirelessDT is to simulate the propagation of wireless signals in a virtual dynamic environment and generate simulated data. For instance, when given the coordinates of a specific location to be observed, WirelessDT is employed to obtain observation results for the corresponding coordinates in a given scene. Consequently, WirelessDT can also be regarded as a generative model, denoted as $\mathcal{G}$.

Let us consider a dataset comprising N training examples represented as $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$. Here, $x^{(i)}$ signifies the characteristic vector of the i^{th} example and $y^{(i)}$ corresponds to the relevant label. On providing an feature vector $x \in \{x^{(1)}, \dots, x^{(n)}\}$ to WirelessDT, it generates an observation $\hat{y}^{(i)}$ based on $x^{(i)}$:

$$\hat{y}^{(i)} = \mathcal{G}(x^{(i)}) \quad (4.3)$$

In practical scenarios, $x^{(i)}$ could represent diverse factors, such as the measured location, among others, and $y^{(i)}$ represents the corresponding predicted observation. the supervised learning can be employed to train the model $\mathcal{G}$ and push $\hat{y}^{(i)}$ close to $y^{(i)}$, enabling more accurate and realistic observation generation by learning from the actual dataset $\mathcal{D}$.

WirelessDT can be parameterised to produce a trainable model $\mathcal{G}_\psi$, where ψ represents the model parameters. Based on the training objective, the model parameters ψ can be segregated into trainable parameters θ and non-trainable parameters ϕ . Within the WirelessDT framework, the trainable parameters θ signify the model parameters is adjusted for an optimal fit. These could include aspects like transmitter position, power, frequency, system gain, as well as material permittivity, and permeability, all of which influence signal propagation but are not readily available from the dataset. Non-trainable parameters ϕ , on the other hand, are pre-set before training begins. Examples include building structures within the scene, furniture type and location, and people count. These parameters can be readily obtainable from available data sources such as building floor plans and pedestrian statistics. The generative model $\mathcal{G}_{\theta,\phi}$, with excellent interpretability and controllability, promoting optimal use of the rich information present within actual datasets. This helps in fine-tuning the model and strategically selecting the training parameters.

4.5.3 Training

4.5.3.1 Objective Function

In order to optimize the model parameters of $\mathcal{G}_\psi$ so that the difference between the model generated data and the real data is as small as possible, this study propose to minimize the [Euclidean Distance \(ED\)](#) and [Jensen-Shannon Divergence \(JS Divergence\)](#) between the generated data and the real data as the training objectives.

ED measure the distance between two points in euclidean space. For the distance between the generated result $\hat{y}^{(i)}$ and the label $y^{(i)}$ with K dimensions, the ED can be formulated by:

$$D_{ED}(\hat{y}^{(i)}, y^{(i)}) = \sqrt{\sum_{k=1}^K (\hat{y}^{(i,k)} - y^{(i,k)})^2} = \|\hat{y}^{(i)} - y^{(i)}\| \quad (4.4)$$

JS Divergence is a symmetric divergence derived from **KL Divergence**. Given two probabilistic distributions p and q , the **JS Divergence** is written as:

$$D_{JS} = \frac{1}{2}D_{KL}(p||h) + \frac{1}{2}D_{KL}(q||h) \quad (4.5)$$

where $h = \frac{1}{2}(p + q)$. For generative models with higher fidelity, the probability distributions of the generated observation have a greater similarity to the probability distributions of the real data, making the JS divergence between the distributions smaller.

To calculate **KL Divergence** D_{KL} , p and q need to be retrieved from the data. Assume that M samples is token from the dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$, and splitted into a set of features $\mathcal{X} = \{x^{(i)}\}_{i=1}^M$ and a set of labels $\mathcal{Y} = \{y^{(i)}\}_{i=1}^M$. Relatively, $\hat{\mathcal{Y}} = \{\hat{y}^{(i)}\}_{i=1}^M$ denotes the set of predictions of the generative model $\mathcal{G}_\Phi$ with given $\mathcal{X}$.

Then the probabilistic distribution of true observation of $\mathcal{Y}$ can be calculated by the integral:

$$q(y) = \int q(y|x)q(x) \quad (4.6)$$

Similarly, the probabilistic distribution of the observation generated from $\hat{\mathcal{Y}}$ is:

$$p(\hat{y}) = \int p(\hat{y}|x)p(x) = \int p(\mathcal{G}(x)|x)p(x) \quad (4.7)$$

The **KL Divergence** can be calculated by:

$$D_{KL}(p||q) = \int p(\hat{y}) \log \frac{p(\hat{y})}{q(y)} \quad (4.8)$$

For a batch of sample $\mathcal{X}, \mathcal{Y}$ with M elements from dataset $\mathcal{D}$. The objective function can be written as minimizing the Euclidean distance for each data point as well as the JS-divergence for overall distribution:

$$\min \arg \underbrace{\frac{1}{M} \sum_{i=1}^M \|\hat{y}^{(i)} - y^{(i)}\|}_{\text{Dissimilarity between data points}} + \underbrace{D_{JS}(p(\hat{y})||q(y))}_{\text{Dissimilarity between distribution}} \quad (4.9)$$

Moreover, β is introduced as the weight of importance between data point similarity and distribution similarity, then the loss which is trying to be minimized is:

$$\mathcal{L} = \frac{1}{M} \sum_{i=1}^M \|\hat{y}^{(i)} - y^{(i)}\| + \beta D_{JS}(p(\hat{y})||q(y)) \quad (4.10)$$

With Equation 4.10, the generated data fidelity can be maximized by minimizing the difference between the data point dissimilarity and the data distribution dissimilarity.

4.5.3.2 Minimizing with Monte Carlo

To solve this minimization, this study adapt Monte Carlo methods. Given a set of inputs $\mathcal{X}$, the corresponding output set $\hat{\mathcal{Y}}$ can be generated by a generative model $\mathcal{G}_\Psi$ that is:

$$\hat{\mathcal{Y}} = \mathcal{G}_\Psi(\mathcal{X}) \quad (4.11)$$

In a practical problem, the boundary of model parameters Ψ might be known, then a set of parameters $\{\Psi^{(l)}\}$ can be sampled within the boundary. Depending on the precondition of the parameter, **Uniform sampling** or **Gaussian sampling** can be used.

- **Uniform sampling:** For the case where the Ψ parameter is completely unknown, uniform sampling which selects samples at fixed intervals in the parameter space for each dimension independently can be used:

$$\Psi_i \sim U(a_i, b_i) \quad (4.12)$$

This approach is suitable for the case where the boundaries are narrow.

- **Gaussian sampling:** In scenarios where the rough values of parameters can be estimated in advance, it is possible to center these estimations in the Gaussian sampling process and specify the amount of variation,

$$\Psi \sim \mathcal{N}(\mu_\Psi, \Sigma_\Psi) \quad (4.13)$$

where μ_Ψ is a high-dimensional vector representing the mean of the rough value and Σ_Ψ is the covariance matrix.

And for each of the $\Psi \in \{\Psi^{(l)}\}$, the same set of input $\mathcal{X}$ is used to generate the prediction $\hat{\mathcal{Y}}_{\Psi^{(l)}}$, where

$$\hat{\mathcal{Y}}_{\Psi^{(l)}} = \mathcal{G}_{\Psi^{(l)}}(\mathcal{X}) \quad (4.14)$$

Then the loss for each of these generated observations is calculated.

$$\mathcal{L}_{\Psi^{(l)}} = \frac{1}{M} \sum_{i=1}^M \|\hat{y}_{\Psi^{(l)}}^{(i)} - y^{(i)}\| + \beta D_{JS}(p(\hat{y}_{\Psi^{(l)}}) || q(y)) \quad (4.15)$$

The set consisting of these loss values $\mathcal{L}_{\Psi^{(l)}}$ can essentially be viewed as a sampling of the loss function under the given boundary conditions. Therefore, minimizing the loss function can be done by finding the parameter $\Psi^{(l)}$ that corresponds to the smallest value in the set $\{\mathcal{L}_{\Psi^{(l)}}\}$ constituted by loss.

4.5.3.3 Parameter Update

Assuming the loss space is smooth, and the parameters $\Psi_{\min}$ that yield the minimal loss have been identified, the model parameters Ψ can be updated using the learning rate α that:

$$\Psi \leftarrow \Psi + \alpha(\Psi_{\min} - \Psi) \quad (4.16)$$

By repeatedly sampling the process of $\mathcal{X}$ and Ψ , the model parameters can be optimized iteratively.

In addition, a loss weight $W_{\mathcal{L}}$ can be defined:

$$W_{\mathcal{L}} = \frac{\mathcal{L}_{\Psi^{(l)}}}{\sum_{l=1}^L \mathcal{L}_{\Psi^{(l)}}} \quad (4.17)$$

where smaller loss contributes to smaller weight. According to the loss weight $W_{\mathcal{L}}$ and the probability being sample $p(\Psi^{(l)})$, the weight sum on all samples to update the parameter Ψ can be performed. that is:

$$\Psi \leftarrow \Psi + \alpha \sum_{l=1}^L p(\Psi^{(l)}) (1 - W_{\mathcal{L}}) (\Psi^{(l)} - \Psi) \quad (4.18)$$

where a smaller loss weight contributes to a larger update weight for parameter update.

To enhance accuracy, narrowing the boundary of Ψ based on previous loss results or increasing the number of samples to improve the density of sampling given boundary conditions. The overall training process is shown as follows:

Algorithm 4 WiTraining algorithm.

Input: Data set $\mathcal{D}$, generative model $\mathcal{G}$, learning rate α , loss importance factor β

Output: Model parameters Ψ .

initialize Ψ

initialize $\mathcal{G}_{\Psi}$

repeat

 samples M data points from dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$ as batch dataset $\mathcal{D}_M$

 split batch dataset $\mathcal{D}_M$ into feature set $\mathcal{X} = \{x^{(i)}\}_{i=1}^M$ and label set $\mathcal{Y} = \{y^{(i)}\}_{i=1}^M$

 sample a set of Ψ from Gaussian distribution $\Psi \sim \mathcal{N}(\mu_{\Psi}, \Sigma_{\Psi})$ as $\{\Psi^{(l)}\}$

for $\Psi^{(l)} \in \{\Psi^{(l)}\}$ **do**

$\hat{\mathcal{Y}}_{\Psi^{(l)}} = \mathcal{G}_{\Psi^{(l)}}(\mathcal{X})$

 ▷ generate prediction set

$\mathcal{L}_{\Psi^{(l)}} = \frac{1}{M} \sum_{i=1}^M \|\hat{y}_{\Psi^{(l)}}^{(i)} - y^{(i)}\| + \beta D_{JS}(p(\hat{y}_{\Psi^{(l)}}) || q(y)), \hat{y}_{\Psi^{(l)}}^{(i)} \in \hat{\mathcal{Y}}_{\Psi^{(l)}}$

 ▷ calculate loss

end for

$W_{\mathcal{L}} = \frac{\mathcal{L}_{\Psi^{(l)}}}{\sum_{l=1}^L \mathcal{L}_{\Psi^{(l)}}}$

 ▷ calculate loss weight

$\Psi \leftarrow \Psi + \alpha \sum_{l=1}^L p(\Psi^{(l)}) (1 - W_{\mathcal{L}}) (\Psi^{(l)} - \Psi)$

 ▷ update parameters

until converge

4.6 Proof-of-Concept Implementation of WiTracing

To implement WiTracing in [UE5](#), which utilizes [GPU RTRT](#) for simulating wireless signal propagation, it is necessary to incorporate the WiTracing rendering pipeline into the engine's source code. This involves implementing the WiTracing module in the upper C++ source code, managing WiTracing parameter passing (from the game thread to the rendering thread), and handling the return of WiTracing results from the rendering thread to the game thread. The wireless [RT](#) process is executed within the [Unreal Shader File \(USF\)](#) and [Unreal Shader Header \(USH\)](#) rendering code.

4.6.1 Extending Render Module for WiTracing

4.6.1.1 WiTracing Renderer API in Game Thread

The *Internal_WiTracing* function, as illustrated in [Code 4.3](#), is an internal method devised to enable the utilization of WiTracing functionality in external projects through [APIs](#). This function accepts several parameters, including a *UWorld* object that represents the game world, arrays of *AWirelessTXs* and *AWirelessRXs* symbolizing [Tx](#)s and [Rx](#)s respectively, and employs them to construct a *WiTracingScene* for rendering.

The *UTextureRenderTarget2D* parameter is a 2D image buffer used to store the results of the WiTracing rendering process. The *FWiTracingResult* array is employed to preserve the ray tracing results for each [Tx-Rx](#) pair. The function also accommodates several optional flags. The *OctahedralProjection* flag governs the projection method used to generate primary tracing rays. The *bDenoised* flag denotes whether [DL](#) denoising should be activated. Finally, the *bPostProcessing* flag determines whether post-processing should be applied to intermediate results.

During its execution, the *Internal_WiTracing* function initially verifies if all necessary components are prepared for WiTracing. Subsequently, it uses [Tx](#)s, [Rx](#)s, and world information to construct the WiTracing scene. Then it invokes the renderer module to perform WiTracing

in the given scene. Upon completion of WiTracing, it extracts results from the raw data and stores them in the *FWiTracingResult* array.

CODE 4.3: Source/Runtime/Renderer/Private/WiTracing/WiTracingRenderBlueprintLibrary.cpp

```
void Internal_WiTracing(
    UWorld* World,
    class UTextureRenderTarget2D* RenderTargetTexture,
    TArray<AWirelessTX*> WirelessTXs,
    TArray<AWirelessRX*> WirelessRXs,
    TArray<FWiTracingResult*> Results,
    bool OctahedralProjection = true,
    bool bDenoised = false,
    bool bPostProcessing = true){
    // Check whether all required components is ready for WiTracing
    bool bReady = RenderTargetTexture && World && WirelessTXs.Num() > 0 && WirelessRXs.Num()
        > 0;

    if (!bReady) { return; }

    // Prepare WiTracingScene for Rendering
    FWiTracingScene* WiTracingScene = new FWiTracingScene(World, WirelessTXs, WirelessRXs,
        OctahedralProjection, bDenoised,
        bPostProcessing);

    // Prepare Canvas and ViewFamily for Rendering
    FTextureRenderTargetResource* RenderTargetResource = RenderTargetTexture->
        GameThread_GetRenderTargetResource();

    FSceneInterface* Scene = World->Scene;
    FSceneViewFamilyContext ViewFamily(FSceneViewFamily::ConstructionValues(
        RenderTargetResource, Scene, FEngineShowFlags
        (ESFIM_Game)).
        SetTime(FGameTime::GetTimeSinceAppStart()));
    ViewFamily.SetScreenPercentageInterface(new FLegacyScreenPercentageDriver(ViewFamily, /*
        GlobalResolutionFraction = */ 1.0f));

    ViewFamily.EngineShowFlags.DistanceCulledPrimitives = true;
    ViewFamily.EngineShowFlags.DisableOcclusionQueries = true;
    ViewFamily.EngineShowFlags.PathTracing = true;
    ViewFamily.EngineShowFlags.RayTracingDebug = false;
    ViewFamily.EngineShowFlags.MotionBlur = false;

    // Create View for each receiver
    TArray<FSceneViewStateReference> ViewStates;
    ViewStates.Init(FSceneViewStateReference(), WiTracingScene->RXSceneInfos.Num());

    uint32 Idx = 0;
    for (auto RX : WiTracingScene->RXSceneInfos) {
```

```

        ViewStates[Idx].Allocate(ViewFamily.GetFeatureLevel());
        FTransform Transform = RX.GetTransform();
        FSceneView* View = CalcSceneView(&ViewFamily, Transform, RenderTargetResource->
                                         GetSizeXY(), ViewStates[Idx].GetReference());

        Idx++;
    }
    FCanvas Canvas(RenderTargetResource, NULL, FGameTime::GetTimeSinceAppStart(), Scene->
                  GetFeatureLevel());

    Canvas.Clear(FLinearColor::Transparent);
    // Call WiTracing Renderer to perform rendering task
    GetRendererModule().WiTracingBeginRenderingViewFamily(&Canvas, &ViewFamily,
                                                         WiTracingScene);

    FlushRenderingCommands();
    // Extract result from scene
    if (WiTracingScene && WiTracingScene->bPostProcessing) {
        Results = FWiTracingResult::ExtractResults(WiTracingScene);
    }
}

```

4.6.1.2 Start WiTracing in Render Thread

The *Internal WiTracing* method initiates the WiTracing rendering pipeline by invoking the *FRendererModule::WiTracingBeginRenderingViewFamily* function (Code 4.4) of the renderer module. This function, which serves as the entry point for WiTracing rendering, takes three arguments: a *Canvas* object, a *ViewFamily* object, and a *WiTracingScene* object.

Initially, it triggers end-of-frame updates to ensure all render proxies are current prior to commencing WiTracing. It then creates an instance of *FWiTracingShadingSceneRenderer* and forwards it to the render thread. Subsequently, it queues a render command dedicated to WiTracing. If render thread polling is activated, the render thread is awakened.

CODE 4.4: Source/Runtime/Renderer/Private/WiTracing/WiTracingSceneRendering.cpp

```

static void WiTracingRenderViewFamily_RenderThread(FRHICmdListImmediate& RHICmdList,
                                                    FWiTracingShadingSceneRenderer* SceneRenderer
                                                    ) {
    SceneRenderer->RenderThreadBegin(RHICmdList);
    FRDGBuilder GraphBuilder(
        RHICmdList,
        RDG_EVENT_NAME("WiTracingSceneRenderer"),

```

```

    FSceneRenderer::GetRDGParalelExecuteFlags(SceneRenderer->FeatureLevel)
);
SceneRenderer->Render(GraphBuilder);
GraphBuilder.Execute();
SceneRenderer->RenderThreadEnd(RHICmdList);
}

void FRendererModule::WiTracingBeginRenderingViewFamily(FCanvas* Canvas, FSceneViewFamily*
ViewFamily, FWiTracingScene* WiTracingScene){
    TRACE_CPUPROFILER_EVENT_SCOPE(
        WiTracingBeginRenderingViewFamily);

    check(Canvas);
    check(ViewFamily);
    check(ViewFamily->Scene);
    UWorld* World = nullptr;
    FScene* const Scene = ViewFamily->Scene->GetRenderScene();
    if (Scene){
        World = Scene->GetWorld();
        if (World){ World->SendAllEndOfFrameUpdates(); }
        Canvas->Flush_GameThread();
        FWiTracingShadingSceneRenderer* SceneRenderer = new FWiTracingShadingSceneRenderer(
            ViewFamily, Canvas->GetHitProxyConsumer());
        SceneRenderer->WiTracingScene = WiTracingScene;
        ENQUEUE_RENDER_COMMAND(FWiTracingCommand) ([SceneRenderer] (FRHICommandListImmediate&
            RHICmdList) {
            WiTracingRenderViewFamily_RenderThread(RHICmdList, SceneRenderer);
            FlushPendingDeleteRHIMResources_RenderThread();
        });
        if (GRenderThreadPollingOn){
            FTaskGraphInterface::Get().WakeNamedThread(ENamedThreads::GetRenderThread());
        }
    }
}

```

4.6.1.3 WiTracing Shading Scene Renderer

Code 4.5 shows the class definition of the *FWiTracingShadingSceneRenderer* which is derived from *FDeferredShadingSceneRenderer*. *FDeferredShadingSceneRenderer* is the built-in class in Unreal Engine that is responsible for the deferred shading rendering pipeline. In the original implementation of the deferred shading rendering pipeline, Unreal Engine combine GPU

RT with classical rasterization in the deferred rendering pipeline for scene rendering. Based on the original implementation, additional methods are implemented to support WiTracing.

CODE 4.5: Source/Runtime/Renderer/Private/WiTracing/WiTracingShadingRenderer.h

```
class FWiTracingShadingSceneRenderer : public FDeferredShadingSceneRenderer {
public:
    FWiTracingShadingSceneRenderer(const FSceneViewFamily* InViewFamily, FHitProxyConsumer*
                                   HitProxyConsumer);

    virtual void Render(FRDGBuilder& GraphBuilder) override;
    bool SetupWiTracingPipelineStates(FRHICmdListImmediate& RHICmdList);
    void WaitForWiTracingScene(FRDGBuilder& GraphBuilder, FRDGBufferRef
                               DynamicGeometryScratchBuffer);

    static void PrepareWiTracing(const FSceneViewFamily& ViewFamily, TArray<
                                FRHIRayTracingShader*>& OutRayGenShaders);

    void RenderWiTracing(FRDGBuilder& GraphBuilder, const TArray<FViewInfo>& ViewInfos,
                        FRDGTextureRef SceneColorOutputTexture);

public:
    FWiTracingScene* WiTracingScene;
};
```

- *PrepareWiTracing* prepare the ray-generating shaders required for WiTracing rendering, provided that the conditions for rendering **RT** effects are met, and store them in the global shader list.
- *SetupWiTracingPipelineState* sets the rendering state of the raytracing, including handling the mesh batch task for **RT**, initializing and copying some shared resources, and creating raytraced lighting data for each view. Finally, if all the settings are completed successfully, then true is returned, indicating that the setup was successful.
- *WaitForWiTracingScene* prepares and synchronizes the wait for WiTracing scenes. Iterate through all views and check if there is a raytracing process that needs to be enabled; if not, the function does nothing. It then adds a new Pass to the graphics builder, waits for the **RT** material binding task to complete, collects and commits all shader parameters, transfers ownership of the **RT** material binding to the command list, and some other setup operations.
- *Render* contains a series of steps from view setup, scene preparation, WiTracing calculation, to post-processing and resource release. The *Render* method in the

deferred rendering pipeline is overridden and embedded in the WiTracing calculation. *Render* first validates the availability and configuration of **RT** for the scene. For each view in the scene, it initializes the relevant parameters and shaders. The scene's primitives are updated and the views are prepared for rendering. System and scene textures are set up, and operations for **GPU**-based scenes, instance culling, and final pipeline state are handled. If **RT** is on, world instances are gathered for a reference view. If conditions allow, WiTracing rendering is performed. The function then attempts to generate a view family texture for post-processing operations, if successful. Debug operations for each view are ended, and a post-render delegate operation is broadcasted. If **RT** is on, the resources are released. Finally, the function concludes by completing a render-finish operation, adding a dispatch hint, queuing scene texture extractions, and resetting prior view info for each view.

- *RenderWiTracing* is responsible for rendering scenes using WiTracing, a wireless signal propagation simulation method. The function accepts three arguments:
 - *GraphBuilder*, which is a reference to a **Render Dependency Graph (RDG)** builder that helps to manage and optimize the execution of **GPU** tasks.
 - *ViewInfos*, an array of views within the scene that provide different perspectives.
 - *SceneColorOutputTexture*, a reference to the texture that will store the final result of the rendering process.

The function first verifies if the platform on which it is running supports path tracing, an essential requirement for WiTracing. If the platform is not supported, the function will exit early. The function then retrieves the number of **Tx** and **Rx** scene information. These refer to the number of points or nodes in the scene that receive or transmit signals, respectively. Next, it prepares buffers to store the final radiance and phase results. The number of these results is equal to the number of **RT-Tx** pairs in the scene. The function then processes each **Rx** view separately, and each view represents a different perspective in the scene. For each **Rx**, the function initializes the WiTracing configuration and extracts **Rx** information. Then, for each **Tx** in the scene, the function performs the WiTracing process. For each **Rx-Tx** pair, the function generates a pair of textures, *RadTex* and *PhaTex*, that will save the

results of the WiTracing process. If the *bPreview* flag is true, the function performs the WiTracing and a composite pass once, and then terminates early. When the *bPostProcessing* flag is enabled, the function performs a WiTracing pass for each *Rx-Tx* pair and begins a parallel accumulation process, which involves aggregating nearby buffer values into a corresponding buffer in the down-scale level, which aggregate the simulation result to smaller dimension. After completing this parallel accumulation, the function extracts data from the final image buffer to floating-point values and stores it in the radiances and phases buffers. If the *bDenoised* flag is set, a denoising process is performed on the final image to reduce noise, thus improving the visual quality of the result. The denoised image replaces the original RadTex image. Lastly, a data readback pass is added to *RDG*, which is responsible for transferring content from *GPU* memory back to *CPU* memory. If post-processing is enabled, a blank pass is added to clear or reset view-specific data for the next frame. The detailed implementation is shown in Code 4.6.

CODE 4.6: Source/Runtime/Renderer/Private/WiTracing/WiTracing.cpp

```
void FWiTracingShadingSceneRenderer::RenderWiTracing(
    FRDGBuilder& GraphBuilder,
    const TArray<FViewInfo>& ViewInfos,
    FRDGTextureRef SceneColorOutputTexture) {
    RDG_GPU_STAT_SCOPE(GraphBuilder, Stat_GPU_WiTracing);
    RDG_EVENT_SCOPE(GraphBuilder, "WiTracing");
    if (!ensureMsgf(FDataDrivenShaderPlatformInfo::GetSupportsPathTracing(Views[0].
                                                                    GetShaderPlatform()), TEXT("Attempting to use
                                                                    WiTracing on unsupported platform."))) {
        return;
    }

    uint32 NumRXs = WiTracingScene->RXSceneInfos.Num();
    uint32 NumTXs = WiTracingScene->TXSceneInfos.Num();
    uint32 NumPairs = NumRXs * NumTXs;
    FRDGBuffer* Radiances = nullptr;
    FRDGBuffer* Phases = nullptr;
    FRDGBufferDesc Desc = FRDGBufferDesc::CreateStructuredDesc(sizeof(float), NumPairs);
    Radiances = GraphBuilder.CreateBuffer(Desc, TEXT("WiTracer.Radiance"));
    Phases = GraphBuilder.CreateBuffer(Desc, TEXT("WiTracer.Phase"));
    bool bPreview = !(WiTracingScene->bPostProcessing);
    for (uint32 RXIdx = 0; RXIdx < NumRXs; RXIdx++) {
        const FViewInfo& View = ViewInfos[RXIdx];
```


[illegible]

```

const uint32 Size = FMath::RoundUpToPowerOfTwo(PrevSize / DownscaleFactor);
const FIntPoint Resolution = FIntPoint(Size, Size);
ParallelAccRadTex.Add(nullptr);
ParallelAccPhaTex.Add(nullptr);
FRDGTextureDesc TextureDesc = FRDGTextureDesc::Create2D(Resolution,
    PF_A32B32G32R32F, FClearValueBinding::None,
    TexCreate_ShaderResource | TexCreate_UAV);
ParallelAccRadTex[Level] = GraphBuilder.CreateTexture(TextureDesc,
    ParallelAccRadTexName, ERDGTextureFlags::None
);
ParallelAccPhaTex[Level] = GraphBuilder.CreateTexture(TextureDesc,
    ParallelAccPhaTexName, ERDGTextureFlags::None
);

// 2.2 perform parallel aggregation
AddWiTracingParallelAccumulatePass(GraphBuilder, View, PrevAccRadTex,
    PrevAccPhaTex, ParallelAccRadTex[Level],
    ParallelAccPhaTex[Level], Resolution);

// 2.3 update previous size and buffer
PrevSize = Size;
PrevAccRadTex = ParallelAccRadTex[Level];
PrevAccPhaTex = ParallelAccPhaTex[Level];
}

// 3 for the last aggregation result extract data from image buffer to float
AddWiTracingAccumulatePass(GraphBuilder, ParallelAccRadTex.Last(),
    ParallelAccPhaTex.Last(), Radiances, Phases,
    RXIdx, TXIdx, NumRXs, NumTXs);
}

if (WiTracingScene->bDenoised) {
    FRDGTextureDesc RadianceTextureDesc = FRDGTextureDesc::Create2D(
        View.ViewRect.Size(),
        PF_A32B32G32R32F,
        FClearValueBinding::None,
        TexCreate_ShaderResource | TexCreate_UAV);
    FRDGTexture* DenoisedRadianceTexture = GraphBuilder.CreateTexture(
        RadianceTextureDesc, TEXT("WiTracer.
        DenoisedRadiance"), ERDGTextureFlags::
        MultiFrame);
    FRHIGPUMask GPUMask = GraphBuilder.RHICmdList.GetGPUMask();
    FDenoiseTexParameters* DenoiseParameters = GraphBuilder.AllocParameters<
        FDenoiseTexParameters>();
    DenoiseParameters->InputTexture = RadTex;
    DenoiseParameters->InputAlbedo = RadTex;

```

```

DenoiseParameters->InputNormal = RadTex;
DenoiseParameters->OutputTexture = DenoisedRadianceTexture;
int DenoiserMode = 1;
GraphBuilder.AddPass(RDG_EVENT_NAME("Wi Tracer Denoiser Plugin"),
                    DenoiseParameters, ERDGPassFlags::Readback, [
                    DenoiseParameters, DenoiserMode, GPUMask] (
                    FRHICommandListImmediate& RHICmdList) {
    GPathTracingDenoiserFunc(RHICmdList,
        DenoiseParameters->InputTexture->GetRHI()->GetTexture2D(),
        DenoiseParameters->InputAlbedo->GetRHI()->GetTexture2D(),
        DenoiseParameters->InputNormal->GetRHI()->GetTexture2D(),
        DenoiseParameters->OutputTexture->GetRHI()->GetTexture2D(),
        GPUMask);
});
RadTex = DenoisedRadianceTexture;
}}}}
AddWiTracingDataReadBackPass(GraphBuilder, Radiances, WiTracingScene, NumTXs*NumRXs);
if (WiTracingScene->bPostProcessing) {AddBlankPass(GraphBuilder, Views[0],
                                                SceneColorOutputTexture);}
}

```

4.6.2 Perform WiTracing in Shaders

4.6.2.1 Shader Binding

To bind the shader code to the Unreal Engine, five *FGlobalShader* classes have been implemented into the UE5's source code, as shown in Code 4.7:

CODE 4.7: Source/Runtime/Renderer/Private/WiTracing/WiTracing.h

```

class FWiTracingRG : public FGlobalShader {...};
class FWiTracingParallelAccumulatorCS : public FGlobalShader {...};
class FWiTracingAccumulatorCS : public FGlobalShader {...};
class FWiTracingCompositorPS : public FGlobalShader {...};
class FWiTracingBlankPS : public FGlobalShader {...};

```

The added shaders are as follows:

- *FWiTracingRG* is a ray generation shader, bound to the *WiTracingMainRG* shading function. It accepts five inputs:
 - *FWiTracingData*: This data structure contains the configuration and parameters necessary for WiTracing.
 - *RaytracingAccelerationStructure*: This is a top-level acceleration structure designed for the RT process, facilitating optimized searches for ray/primitive intersections during rendering.
 - *StructuredBuffer<FWiTracingRx>* and *SceneReceiverCount*: These inputs encapsulate the parameters for *Txs* within the scene. The *StructuredBuffer* object stores a collection of *Rx* data, while *SceneReceiverCount* provides the number of *Txs*.
 - *FViewUniformShaderParameters*: These parameters encompass the position and orientation data for the *Rx* in the scene.
 - *RWRadianceTexture* and *RWPhaseTexture*: These outputs hold the WiTracing results, with *RWRadianceTexture* and *RWPhaseTexture* respectively storing the radiance and phase of the signal at each pixel. These data form the basis for creating the final visual representation of the propagation of wireless signals within the scene.
- *FWiTracingParallelAccumulatorCS* is a compute shader, binding to *ParallelAccumulateMainCS* shading function, that aggregates WiTracing results from *FWiTracingRG*. This shader is designed to stack multiple times, and with each iteration, it reduces the size of the output by a factor of 2.
- *FWiTracingAccumulatorCS* is a nonparallel version of *FWiTracingParallelAccumulatorCS*, binding to *AccumulateMainCS*. It is executed after the last *FWiTracingParallelAccumulatorCS*, aggregating the radiance and phase results from a 2D image texture buffer into a float.
- *FWiTracingCompositorPS* is the pixel shader, binding to *CompositeMain* for the intermediate output of the WiTracing process. It maps the radiance of each pixel to a

color corresponding to the signal strength for visualization purposes. *FWiTracingBlankPS* is binding to *BlankMain* function that is responsible to clean and reset the render target for visuailization.

4.6.2.2 WiTracing Shader

The *WiTracingMainRG* and *WiTracingKernel* functions form the core of the implementation of the ray generation shader. Here, *WiTracingMainRG* governs the boundary conditions during the tracing process, while *WiTracingKernel* defines the manner in which the ray propagates through the scene. The detailed implementation of *WiTracingMainRG* is presented in Code 4.8.

CODE 4.8: Source/Shaders/Private/WiTracing/WiTracing.usf

```
RAY_TRACING_ENTRY_RAYGEN(WiTracingMainRG) {
    const uint2 DispatchIdx = DispatchRaysIndex().xy;
    const uint2 DispatchDim = DispatchRaysDimensions().xy;
    RWRadianceTexture[DispatchIdx] = 0;
    RWPhaseTexture[DispatchIdx] = 0;
    for (uint Iteration = 0; Iteration < MaxSamples; Iteration++){
        FPathState PathState = CreatePathState(DispatchIdx, DispatchDim, Iteration +
                                                TemporalSeed);
        for (int Bounce = 0; Bounce <= MaxBounces; Bounce++) {
            if (!WiTracingKernel(PathState, Bounce)) {break;}
        }
        if (!AcceptPacket(mWTodBm(PathState.Radiance), DispatchIdx, Iteration + TemporalSeed))
            {break;}

        PathState.SaveToGlobal(Iteration);
    }
    RWRadianceTexture[DispatchIdx] /= float(MaxSamples);
}
```

The *WiTracingMainRG* function can be delineated into five distinct stages:

- **Initialization:** As the first step, *WiTracingMainRG* initializes the storage for Radiance and Phase for each pixel, denoted by *DispatchIndex*, in the Radiance Texture

(*RWRadianceTexture*) and Phase Texture (*RWPhaseTexture*). These textures are designed to accommodate the results of the WiTracing process.

- **WiTracing Iteration:** This stage commences WiTracing for each sample, up to a maximum limit specified by *MaxSamples*. To do this, *WiTracingMainRG* initializes the *PathState* object for each sample with parameters such as *DispatchIndex*, *DispatchDimensions* and a temporal seed based on the iteration number.
- **WiTracing:** This phase involves iterative tracking of the signal (or ray) for a set number of bounces, as specified by *MaxBounces*. During each bounce, the *WiTracingKernel* function calculates the subsequent bounce or hit. The loop persists until any of the following conditions materializes: reaching the maximum number of bounces, missing a hit point, having no throughput, or reaching the maximum traced distance.
- **Packet Validation:** At this juncture, a packet (or signal) is accepted if it passes the validation performed by the *AcceptPacket* function, which verifies the strength of the signal. Signals deemed too weak to be detected are discarded and do not contribute to the final accumulation.
- **Sample Averaging:** In the final stage, *WiTracingMainRG* averages the results across the number of samples. By dividing the accumulated radiance for each *DispatcherIndex* in the *RWRadianceTexture* by the total number of samples (*MaxSamples*), it determines the average radiance value for each pixel.

Code 4.9 is the implementation of *WiTracingKernel*, which is responsible for generating and propagating rays, and then collecting and processing the data (such as radiance, distance, etc.) after the ray has interacted with the scene.

CODE 4.9: Source/Shaders/Private/WiTracing/WiTracingCore.usf

```
bool WiTracingKernel(inout FPathState PathState, int Bounce) {
    const bool bIsPrimaryRay = Bounce == 0;
    const bool bIsLastBounce = Bounce == MaxBounces;
    FPathTracingPayload Payload = TraceTransparentWi(PathState, bIsPrimaryRay, bIsLastBounce)
                                ;
    if (Payload.IsMiss()) {return false;}
    PathState.AccumulateTracedDistance(Payload.HitT); {
```

```

Payload.WorldNormal = AdjustShadingNormal(Payload.WorldNormal, Payload.WorldGeoNormal,
                                           PathState.Ray.Direction);

Payload.WorldSmoothNormal = AdjustShadingNormal(Payload.WorldSmoothNormal, Payload.
                                                WorldGeoNormal, PathState.Ray.Direction);
}

if (ApproximateCaustics){AdjustMaterialRoughness(Payload, PathState.PathRoughness);}

float4 RandSample = PathState.GenerateRandomSample4D();

float WiPickingCdfSum = 0;
float WiPickingCdf[WI_TRACING_TRANSMITTER_COUNT_MAXIMUM];

const float3 WorldPos = Payload.WorldPos;
const float3 TranslatedWorldPos = Payload.WorldPos + LWCHackToFloat(PrimaryView.
                                                                    PreviewTranslation); // RT_LWC_TODO

for (uint TransmitterId = 0; TransmitterId < SceneTransmitterCount; ++TransmitterId){
    WiPickingCdfSum += EstimateWi(TransmitterId, TranslatedWorldPos);
    WiPickingCdf[TransmitterId] = WiPickingCdfSum;
}

if (WiPickingCdfSum > 0){
    uint TransmitterId;
    float WiPickPdf = 0;
    SelectWi(RandSample.x * WiPickingCdfSum, SceneTransmitterCount, WiPickingCdf,
            TransmitterId, WiPickPdf);

    WiPickPdf /= WiPickingCdfSum;
    FWISample WiSample = SampleWi(TransmitterId, RandSample.yz, TranslatedWorldPos);
    WiSample.RadianceOverPdf /= WiPickPdf;
    WiSample.Pdf *= WiPickPdf;
    if (WiSample.Pdf > 0){
        const float SignedPositionBias = Payload.IsMaterialTransmissive() ? sign(dot(
                                                                    Payload.WorldGeoNormal, WiSample.
                                                                    Direction)) : 1.0;

        const bool bEnableShadowTermFix = true;
        RayDesc WiRay;
        WiRay.Origin = WorldPos;
        WiRay.TMin = 0;
        WiRay.Direction = WiSample.Direction;
        WiRay.TMax = WiSample.Distance;
        ApplyRayBias(WiRay, Payload.HitT, SignedPositionBias * Payload.WorldGeoNormal,
                    bEnableShadowTermFix);

        float AvgRoughness = ApproximateCaustics ? GetAverageRoughness(Payload) : 0.0;
        WiSample.RadianceOverPdf *= TraceTransparentVisibilityWi(WiRay, PathState.
                                                                    PixelIndex, AvgRoughness);

        if (any(WiSample.RadianceOverPdf > 0)){

```

```

    FMaterialEval MaterialEval = EvalMaterial(PathState.Ray.Direction, WiSample.
                                                Direction, Payload, false);

    float3 Contrib = PathState.PathThroughput * WiSample.RadianceOverPdf *
                                                MaterialEval.Pdf * MaterialEval.Weight ;

    Contrib /= MaterialEval.Pdf;
    Contrib *= ReflectionCoef;
    PathState.AccumulateRadiance(Contrib, WiSample.Distance, GetWavelength(
                                                TransmitterId));

}}}

FMaterialSample MaterialSample = SampleMaterial(PathState.Ray.Direction, Payload,
                                                RandSample, false);

if (MaterialSample.Pdf < 0 || asuint(MaterialSample.Pdf) > 0x7F800000){
    PathState.Radiance = 0;
    return false;
}

if (!(MaterialSample.Pdf > 0)){return false;}

float3 NextPathThroughput = PathState.PathThroughput * MaterialSample.Weight;
if (!any(NextPathThroughput > 0)){return false;}

float ContinuationProb = sqrt(saturate(max(NextPathThroughput.x, max(NextPathThroughput.y
                                                                    , NextPathThroughput.z)) / max(PathState.
                                                                    PathThroughput.x, max(PathState.
                                                                    PathThroughput.y, PathState.PathThroughput.z)
                                                                    )));

if (ContinuationProb < 1) {
    float RussianRouletteRand = RandSample.w; // SampleMaterial does not use this value at
                                                the moment

    if (RussianRouletteRand >= ContinuationProb){return false;}

    PathState.PathThroughput = NextPathThroughput / ContinuationProb;
} else {
    PathState.PathThroughput = NextPathThroughput;
}

PathState.Ray.Origin = Payload.WorldPos;
PathState.Ray.Direction = MaterialSample.Direction;
PathState.Ray.TMin = 0;
PathState.Ray.TMax = POSITIVE_INFINITY;
ApplyRayBias(PathState.Ray, Payload.HitT, MaterialSample.PositionBiasSign * Payload.
                                                WorldGeoNormal);

PathState.PathRoughness = max(PathState.PathRoughness, MaterialSample.Roughness); {
    const bool bUseMIS = WiPickingCdfSum > 0;
    const float ShadowRayRoughness = ApproximateCaustics ? PathState.PathRoughness : 0.0;
    for (uint TransmitterId = 0; TransmitterId < SceneTransmitterCount; ++TransmitterId) {
        RayDesc TranslatedRay = PathState.Ray;

```



```

TranslatedRay.Origin += LWCHackToFloat(PrimaryView.PreViewTranslation); //
                                RT_LWC_TODO
FWiHit TraceHit = TraceTransmitter(TranslatedRay, TransmitterId);
if (TraceHit.IsMiss()) {continue;}
float3 Contrib = PathState.PathThroughput * TraceHit.Radiance;
if (bUseMIS) {
    float PreviousCdfValue = 0.0;
    BRANCH if (TransmitterId > 0) {PreviousCdfValue = WiPickingCdf[TransmitterId -
                                1];}

    float WiPickPdf = (WiPickingCdf[TransmitterId] - PreviousCdfValue) /
                                WiPickingCdfSum;

    Contrib *= MISWeightRobust(MaterialSample.Pdf, TraceHit.Pdf * WiPickPdf); //
                                TraceHit.Pdf is always 1

    Contrib *= ReflectionCoef;
}
if (any(Contrib > 0)){
    RayDesc WiRay = PathState.Ray;
    WiRay.TMax = TraceHit.HitT;
    Contrib *= TraceTransparentVisibilityWi(WiRay, PathState.PixelIndex,
                                ShadowRayRoughness);

    const float TransmitterDistance = sqrt(Pow2(TraceHit.HitT) + Pow2(GetRadius(
                                TransmitterId)));

    PathState.AccumulateRadiance(Contrib, TransmitterDistance, GetWavelength(
                                TransmitterId));
}}}
PathState.RayCone = Payload.RayCone;
return true;
}

```

The *WiTracingKernel* function initiates the **RT** process employing the *TraceTransparentWi* function, which traces the ray's path within the scene. This function penetrates the geometry of the scene, calculating the path throughput until it reaches a predefined hit point. As an any-hit shader, it modifies the final hit payload and determines the subsequent tracing behavior after each hit, deciding whether to return the current hit or continue tracing the ray. To emulate the penetration of a wireless signal through a geometric body, *TraceTransparentWi* samples the probability of accepting a hit based on the current accumulated throughput. If the ray does not hit anything, the function returns false and halts execution.

Next, *WiTracingKernel* generates a random sample and computes [Cumulative Distribution Function \(CDF\)](#) for the radiance of each [Rx](#) within the scene. If the total sum of CDF exceeds 0, it selects a [Rx](#) based on the random sample and calculates a sample. Subsequently, it launches a new ray from the payload's world position toward the selected wireless [Rx](#) direction. This new ray, referred to as the shadow ray, is traced within the scene. If the shadow ray is found to carry radiance, this radiance is accumulated within the *PathState*.

The function then refreshes the ray's origin to match the payload's world position, and the ray's direction is adjusted to align with the direction of the sampled material according to [BRDF](#). *WiTracingKernel* then repeats the tracing process for each [Rx](#) within the scene. If it identifies a hit, it computes the hit's radiance contribution. If this contribution is significant (greater than 0), it accumulates this radiance within the *PathState*. Finally, the function updates the ray cone of the *PathState* to match the payload's ray cone and returns true, ready for the next bounce.

4.6.2.3 Data Aggregation Shaders

To aggregate the WiTracing result, two compute shaders *ParallelAccumulateMainCS* (Code 4.10) and *AccumulateMainCS* (Code 4.11) are designed for aggregating 2D image texture buffer to float type.

CODE 4.10: Source/Shaders/Private/WiTracing/WiTracingParallelAccumulatingCS.usf

```
void ParallelAccumulateMainCS(uint2 DispatchThreadId : SV_DispatchThreadID) {
    uint2 InputDim;
    uint2 OutputDim;
    RadianceTexture.GetDimensions(InputDim[0], InputDim[1]);
    RWParallelAccumulatedRadianceTexture.GetDimensions(OutputDim[0], OutputDim[1]);
    const uint2 OutputIndex = DispatchThreadId;
    const uint DownscaleFactor = InputDim / OutputDim;
    float Radiance = 0;
    float Phase = 0;
    float NumSamples = 0;
    bool bFirst = true;
    for (uint X = 0; X < DownscaleFactor; X++) {
        for (uint Y = 0; Y < DownscaleFactor; Y++) {
            const uint2 InputIndex = OutputIndex * DownscaleFactor + uint2(X, Y);
```

```

if (InputIndex.x < InputDim.x && InputIndex.y < InputDim.y) {
    const float R1 = RadianceTexture[InputIndex];
    const float P1 = PhaseTexture[InputIndex];
    if (any(R1 > 0)) {
        NumSamples++;
        if (bFirst) {
            Radiance = R1;
            Phase = P1;
            bFirst = false;
        } else {
            const float R2 = Radiance;
            const float P2 = Phase;
            Radiance = RadianceMax(R1, R2);
            if (Radiance == R1) { Phase = P1; }
            else Phase = P2; }
        }
    }
    RWParallelAccumulatedRadianceTexture[OutputIndex] = Radiance;
    RWParallelAccumulatedPhaseTexture[OutputIndex] = Phase;
}

```

First, *ParallelAccumulateMainCS* retrieves the dimensions of the *RadianceTexture* and *RWParallelAccumulatedRadianceTexture* textures. It then calculates a *DownscaleFactor* by dividing the dimensions of the input texture by the output texture. This factor is later used to loop through *RadianceTexture*. The function then initializes radiance, phase and *NumberSamples* (which counts the radiance values greater than 0) to 0. Then, in a nested loop, it loops through each pixel in a block of size *DownscaleFactor* * *DownscaleFactor* of *RadianceTexture*. It checks if the radiance value for each pixel is greater than 0. For the first pixel with radiance value greater than 0, it assigns its radiance and phase to *Radiance* and *Phase*. For the next pixels with a radiance value greater than 0, it compares the radiance of this pixel (*R1*) with the accumulated radiance (*R2*). If *R1* is greater, it updates *Radiance* and *Phase* with *R1* and *P1* respectively. If *R1* is not greater, it keeps the previous phase value. Finally, it writes the accumulated radiance and phase to the *RWParallelAccumulatedRadianceTexture* and *RWParallelAccumulatedPhaseTexture* at the *OutputIndex* position.

CODE 4.11: Source/Shaders/Private/WiTracing/WiTracingParallelAccumulatingCS.usf

```

void AccumulateMainCS(uint2 DispatchThreadId : SV_DispatchThreadID) {
    uint2 Dim;
    RadianceTexture.GetDimensions(Dim[0], Dim[1]);
    float Radiance = 0;
    float Phase = 0;
    float NumSamples = 0;
    bool bFirst = true;
    for (uint X = 0; X < Dim.x; X++) {
        for (uint Y = 0; Y < Dim.y; Y++) {
            const uint2 Index = uint2(X, Y);
            const float R1 = RadianceTexture[Index];
            const float P1 = PhaseTexture[Index];
            if (any(R1 > 0)) {
                NumSamples++;
                if (bFirst) {
                    Radiance = R1;
                    Phase = P1;
                    bFirst = false;
                } else {
                    const float R2 = Radiance;
                    const float P2 = Phase;
                    RadianceSum(R1, P1, R2, P2, Radiance, Phase);
                }
            }
        }
    }
    uint RadIdx = TXIdx + RXIdx * NumTXs;
    RWRadiance[RadIdx] = Radiance / NumSamples;
    RWPhase[0] = Phase;
}

```

Similarly to the parallel version, *AccumulateMainCS* first retrieves the dimensions of the *RadianceTexture* texture. It then initializes *Radiance*, *Phase*, and *NumSamples* to 0. The function then loops through each pixel in the *RadianceTexture* and *PhaseTexture*. It checks if the radiance value for each pixel is greater than 0. For the first pixel with radiance value greater than 0, it assigns its radiance and phase to *Radiance* and *Phase*. For the next pixels with a radiance value greater than 0, it calls a function named *RadianceSum* with the current radiance and phase values (*R1*, *P1*) and the accumulated radiance and phase (*R2*, *P2*). *RadianceSum* is a wave sum algorithm for accumulated sine wave signals with different magnitudes and

phases. Finally, it writes the average radiance and phase to the *RWRadiance* and *RWPhase* arrays at the calculated index.

4.6.2.4 False Color Shader

To enhance the visualization of intermediate WiTracing results, a technique known as false-color mapping has been implemented, as depicted in Code 4.12. In this mapping scheme, a distinct color is assigned to each degree of radiance. For values residing between two thresholds, a linear interpolation mechanism has been devised. This mechanism seamlessly maps values situated between the upper and lower bounds to an appropriate color representation. Such an approach improves the granularity of the visualization, making it a more effective tool for understanding the WiTracing process.

CODE 4.12: The color mapping for visualize signal strength.

```
#define GRADIENT_RESOLUTION 13
float3 ColorMap(float Radiance) {
    const float3 ColorSpectrum[GRADIENT_RESOLUTION] = {
        float3(0, 0, 0), // black -120 dBm, 0.000000000001mW
        float3(0.5, 0, 0.5), // indigo -110 dBm, 0.000000000001mW
        float3(1, 0, 1), // purple -100 dBm, 0.000000000001mW
        float3(0.5, 0, 1), // bluepurple -90 dBm, 0.000000000001mW
        float3(0, 0, 1), // blue -80 dBm, 0.000000000001mW
        float3(0, 0.5, 0.5), // greenblue -70 dBm, // SampleMaterial does not use this
                                value at
the moment 0.00000001mW
        float3(0, 1, 0), // green -60 dBm, 0.0000001mW
        float3(0.5, 1, 0), // yellowgreen -50 dBm, 0.000001mW
        float3(1, 1, 0), // yellow -40 dBm, 0.0001mW
        float3(1, 0.5, 0), // redyellow -30 dBm, 0.001mW
        float3(1, 0, 0), // red -20 dBm, 0.01mW
        float3(1, 0.5, 0.5), // white-red -10 dBm, 0.1mW
        float3(1, 1, 1) // white 0 dBm, 1mW
    };
    const float Gradient[GRADIENT_RESOLUTION] = {0.0000000000001, 0.000000000001, 0.00000000001,
                                                    0.0000000001, 0.00000001, 0.0000001, 0.000001,
                                                    0.000001, 0.0001, 0.001, 0.01, 0.1, 1};

    float3 OutColor = 0;
    if (Radiance < Gradient[0]) {
```

```

    OutColor = ColorSpectrum[0];
} else if (Radiance > Gradient[GRADIENT_RESOLUTION - 1]) {
    OutColor = ColorSpectrum[GRADIENT_RESOLUTION - 1];
} else {
    for (int Index = 0; Index < GRADIENT_RESOLUTION - 1; Index++) {
        const float Lower = Gradient[Index];
        const float Upper = Gradient[Index + 1];
        if (Radiance <= Upper && Radiance > Lower) {
            const float BlendFactor = (Radiance - Lower) / (Upper - Lower); // normalize
            OutColor = lerp(ColorSpectrum[Index], ColorSpectrum[Index+1], BlendFactor);
            break;}}}
return OutColor;
}

```

4.7 Summary

This chapter introduces the core technology of WirelessDT to realize the real-time simulation of wireless signals: GPU Real-Time WiTracing. In order to achieve the ultimate simulation performance to meet the needs of real-time simulation in digital twins, this study is the first to introduce the **RTRT** rendering pipeline in the rendering engine to the simulation of wireless signals and use the engine to simulate the physical behavior of wireless signals propagation in a three-dimensional dynamic scene. To achieve this goal, this study develops a specialized **RT** rendering pipeline, WiTracing, dedicated to wireless signal propagation simulation by extending the rendering module of an open-source game engine. In addition, this chapter presents a machine learning-based algorithm to improve the fidelity of wireless signal simulation by learning from real data. The algorithm treats the digital twin as a complete generative model and evaluates the quality of data generation by comparing the dissimilarity of the generated data with the real data. Since the digital twin is a parametric machine learning model, the algorithm can learn the model parameters and thus maximize the simulation accuracy by minimizing the differences in data points and distributions between the generated and real data. By effectively combining real-time simulation technology and machine learning technology, the WirelessDT system performs well regarding data generation quality, efficiency, and fidelity. This establishes a solid foundation for the application of WirelessDT in real scenarios.

CHAPTER 5

Experimental Evaluation of WirelessDT

This chapter presents a thorough evaluation of WirelessDT in three key aspects: fidelity, scalability, and practicality. Fidelity gauges the degree of congruity between the data simulated by WirelessDT and the actual empirical data. Scalability assesses the operational efficacy of WirelessDT, considering its capacity to manage large-scale, real-time simulations effectively. Practicality, on the other hand, examines WirelessDT's performance in real-world settings. Initially, the fidelity of WirelessDT is determined by calibrating it with actual datasets to verify data congruity. Subsequently, WirelessDT is benchmarked against commonly used [RT](#)-based simulation systems to demonstrate its superior performance. Finally, WirelessDT's usability is assessed by employing it to evaluate a real-world localization software, emphasizing its practical implications. This chapter's extension is presented in [Chapter 4](#), along with a paper submitted to Transactions on Mobile Computing.

5.1 Introduction

Simulations of wireless signal propagation are often grounded in idealized scenarios. While these models provide critical insight into wireless signal behavior and present an intuitive comprehension of signal propagation, they fail to fully represent the complexity of real-world conditions. Consequently, this disconnect can result in inconsistencies between simulated results and actual outcomes, which underscores the intrinsic limitations of these simulations in fully encapsulating the multifaceted aspects of real-world situations.

In the case of WirelessDT, concerted efforts have been made to incorporate diverse environmental parameters, aiming to approximate real-world scenarios as closely as possible.

However, it is crucial to verify the accuracy of these simulated results through comparison with real-world data. The hypothesis is that if data generated by WirelessDT aligns with real-world measurements, it strongly supports WirelessDT's proficiency in wireless signal modeling and confirms the fidelity of its internal model to real-world phenomena. Based on this hypothesis, the preliminary part of this chapter involves calibrating WirelessDT using a publicly accessible dataset, followed by a validation of the model's precision through a comparative analysis of simulated and real-world data.

The propagation of wireless signals is significantly influenced by surrounding environmental conditions. Interferences, such as pedestrians moving within the environment, can notably affect the received signal, particularly in specific wireless application contexts like indoor localization. To enhance the performance of such wireless systems, exploring the system's responses within dynamic, noise-intensive environments is crucial. In this context, WirelessDT, as a tool for wireless signal research, is equipped to emulate the dynamism of signals. This is achieved by leveraging [GPU RTRT](#) rendering techniques to facilitate the real-time simulation of wireless signals. The real-time efficacy of WirelessDT is validated through a comparative study with two widely used [RT](#)-based simulation tools. Additionally, experiments are conducted to evaluate WirelessDT's real-time performance from multiple perspectives thoroughly.

A real-time pedestrian localization application has been established as a pragmatic use case to evaluate the practicability of WirelessDT. Initially, two conventional localization algorithms were integrated into a mobile application, creating a virtual scene to yield real-time simulation data. These data were then supplied to the mobile positioning service as a virtual sensing data source for real-time testing, enabled by the WirelessDT digital twin architecture. Within this setting, WirelessDT virtualizes the localization scenario, allowing the localization device to interact with the virtual environment for real-time localization. This system acts as a surrogate for an accurate location scenario, facilitating the testing of wireless applications without physical constraints in a cost-effective, efficient, and flexible manner.

5.2 Fidelity Evaluation

5.2.1 Experimental Dataset

To evaluate the fidelity of WirelessDT, this study chose "IPIN 2020 Indoor Localization Competition: Track 3 Smartphone-based (off-site)" data (Torres-Sospedra *et al.*, 2020) from Indoor Positioning and Indoor Navigation (IPIN) dataset as the public real-world data. This dataset is data collected on smartphones in Biblioteca Universitat Jaume I (UJI Library).



FIGURE 5.1: Biblioteca Universitat Jaume I (UJI Library).

This dataset comprises comprehensive physical measurements captured by standard smart-phone sensors, including Wi-Fi, accelerometer, gyroscope, magnetometer, barometer, light sensor, proximity sensor, [AHRS](#), and [Global Navigation Satellite System \(GNSS\)](#). It also includes ground-truth reference positions and detailed indoor maps as shown in Figure 5.2. this study chose this dataset due to its high-quality indoor maps, which enable us to accurately reconstruct the virtual representation of the physical building.



FIGURE 5.2: Indoor maps of UJI library from level 1 to 5.

The data collection process for this dataset was structured into five distinct phases.

- Phase I: During the initial phase of data acquisition, the researcher stood stationary in the reference position and held the mobile device in front of his chest. This position served to calibrate the device's sensors through a series of randomized movements. This preliminary calibration process was carried out for an approximate duration of 30 seconds. Upon completion, the researcher utilized the signal acquisition application to validate the accurate recording of all sensor readings.

- Phase II: Then the researcher held the mobile device and remained static for around 30 seconds.
- Phase III: In the third phase, the researcher embarked on a linear path between reference points, maintaining a consistent walking pace. Throughout this process, the mobile device was held in a stable position in front of the researcher's chest. As the researcher passed each reference point, a timer integrated into the mobile application was triggered, documenting the timestamp of each pass.
- Phase IV: Upon reaching the terminal reference point, the researcher positioned himself in the direction of return, retaining the relative position between them and the mobile device. This stationary phase, similar to the calibration phase, persisted for 30 seconds.
- Phase V: Following the aforementioned waiting period, the researcher retraced the course from the terminal reference point back towards the initial position, progressing from one reference point to the next. The researcher remained constant speed during the movement and meticulously recorded the time to pass each reference point.

Fig. 5.3 shows the accelerometer readings for one acquisition record in the dataset. The x, y, and z accelerations represent the acceleration in each of the three dimensions of the accelerometer, and the gray vertical lines represent timestamps as the researcher passes through the reference position. By observing the amplitude of fluctuations in the accelerometer readings, the five phases of the signal acquisition process can be easily distinguished.

5.2.2 Environmental Modeling

In pursuit of generating data close to the [IPIN](#) dataset, the initial step involved modeling the [UJI Library](#). This process was facilitated by the indoor map of [UJI Library](#) provided within the dataset. Previous attempts to model the [UJI Library](#) have been made by [Gauxachs Sanz \(2021\)](#) and [Varea García \(2020\)](#), with their works rendering more intricate architectural details. Merging these studies with the interior maps provided in the dataset enabled the design of a detailed model of [UJI Library](#). Additionally, Google Maps Street View and interior photos released by [UJI Library](#) were used to compare the modeled interior structure and adjust the

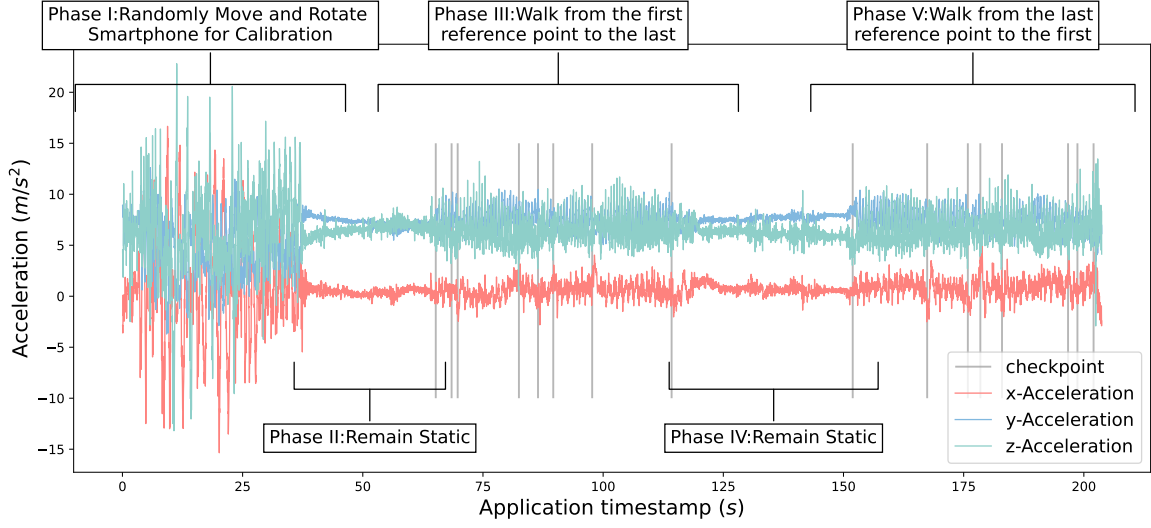


FIGURE 5.3: Data collection process for IPIN dataset.

details. Specifically, floors 2 to 5 were selected as exemplars for modeling, and experiments were based on them as shown in Figure 5.4.

Figure 5.5 displays the comparison between the virtual and the real scene. The exact layout of the shelf is found from the previous work [Varea García \(2020\)](#). Following this, the bookshelf model is created based on online photos of [UJI Library](#) and placed within the virtual scene.

5.2.3 Feature Engineering

5.2.3.1 Data Interpolation

Figure 5.6 is the time of the acquired WiFi signal versus the time the researcher passes through the reference point. The acceleration amplitude a_{acc} is a sum of 3D acceleration vectors, which can be given by:

$$a_{acc} = \sqrt{a_{acc_x}^2 + a_{acc_y}^2 + a_{acc_z}^2} \quad (5.1)$$

The vertical line corresponds to the time that the researcher passed the reference point. The cross symbols correspond to the time points of the wifi measurements. As can be observed in

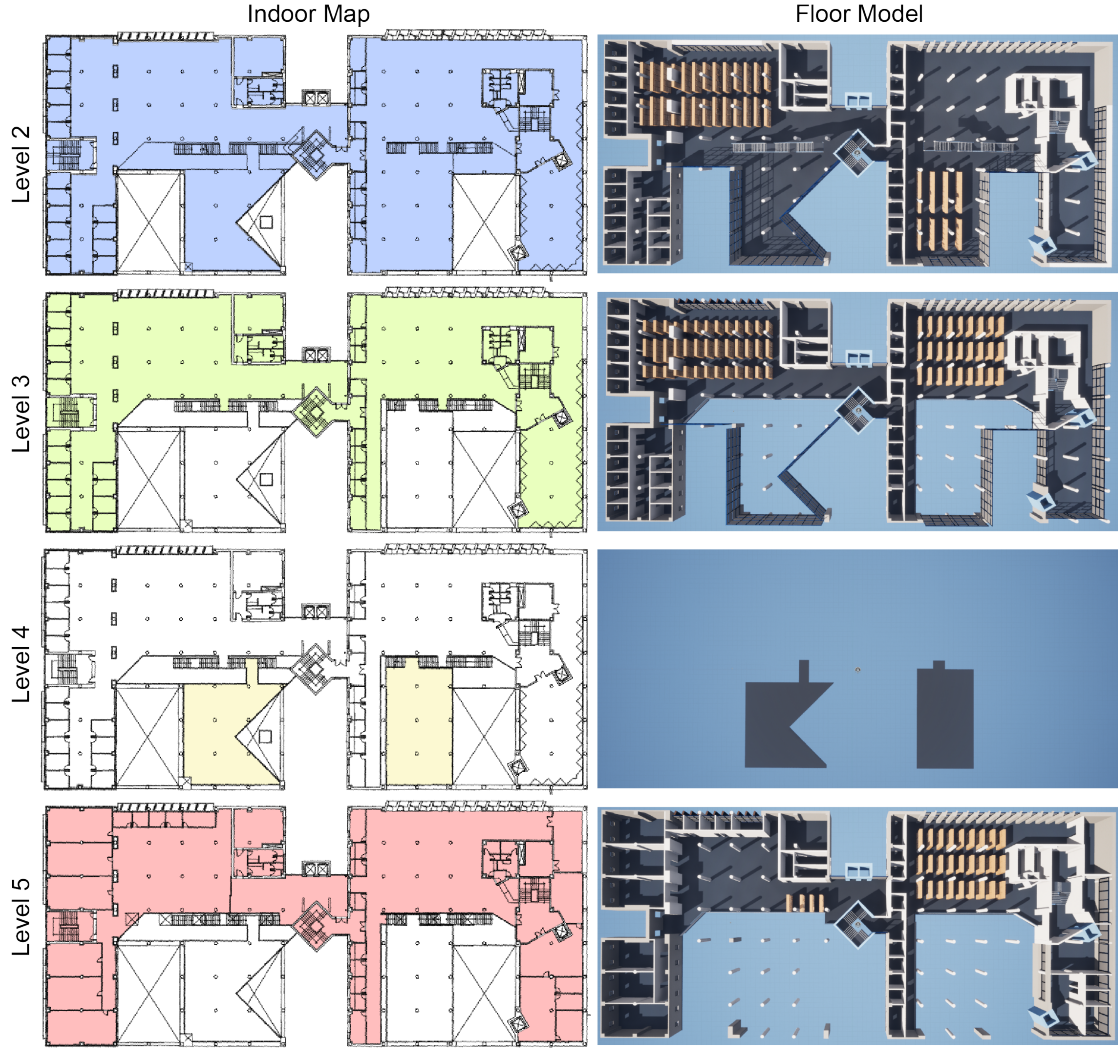


FIGURE 5.4: Building model of UJI library from level 2 to 5.

the figure, there is a fixed interval between the measurements of WiFi by the mobile device. This interval is not affected by the movement of the researcher. Secondly, the time point of the WiFi measurement does not overlap with the time point when the user passes the reference location, i.e. the location of the WiFi measurement is unknown in the original data. Therefore, to better calibrate WirelessDT using the [IPIN](#) dataset, the position of the researcher at the moment of measurement by Wi-Fi.

According to the description file of the dataset, the researcher moves at a constant speed between the reference points. Thus, it can be assumed that the movement is at a constant speed between two consecutive reference points. Given the known positions and times of the

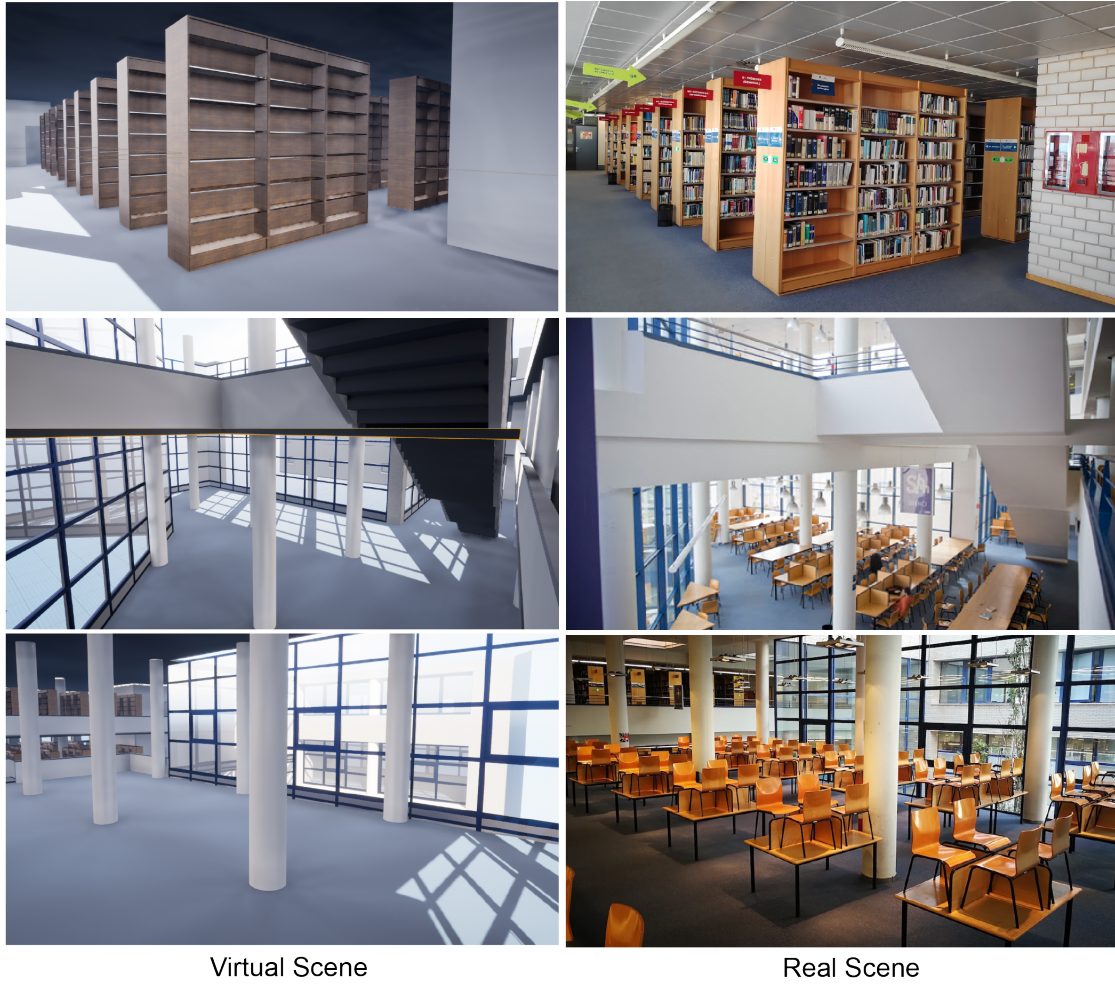


FIGURE 5.5: Comparison of virtual scene and real scene.

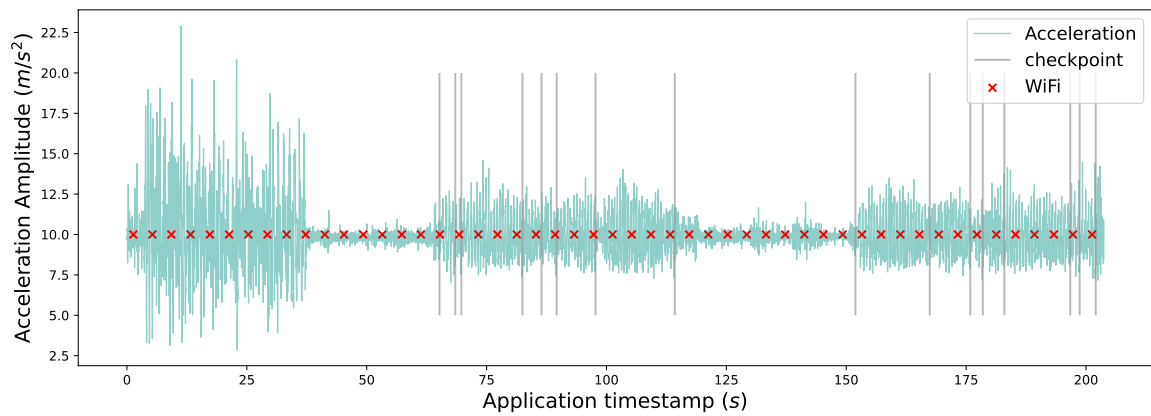


FIGURE 5.6: WiFi measurement in IPIN dataset.

reference points before and after, the position at any moment between the reference points can be obtained through linear interpolation:

$$\begin{aligned} x &= x_1 + \frac{x_2 - x_1}{t_2 - t_1} \\ y &= y_1 + \frac{y_2 - y_1}{t_2 - t_1} \end{aligned} \quad (5.2)$$

Where x, y represent the coordinates and t the timestamp, this method enables the determination of the position coordinates of WiFi measurements at any moment during the researcher's movement. Figure 5.7 illustrates one of the interpolation results of the WiFi measurement. The red dot represents the position of the reference location, while the purple cross indicates the interpolation result of the WiFi measurement.



FIGURE 5.7: Interpolation of WiFi measurement position for single trajectory.

Interpolation of all trajectories from the [IPIN](#) dataset has resulted in the creation of a new dataset, featuring coordinates corresponding to WiFi measurements. The Figure 5.8 illustrates an interpolation outcome for a specific [Media Access Control Address \(MAC address\)](#) on level 2. Signal strength variance is depicted using a color map, where red represents a strong measurement and purple indicates a weak measurement. Plotting all the measurements corresponding to a single [MAC address](#) leads to the generation of a signal strength distribution map.

In the figure, it is discernible that the signal is predominantly concentrated in the map's center. Starting from this area of robust signal strength, the signal progressively decreases outward. The most attenuated signal is located in the upper right corner of the map. These observations are consistent with the signal decay model, which postulates that the signal deteriorates as the distance covered increases. Signal loss is notably more significant when penetrating walls.

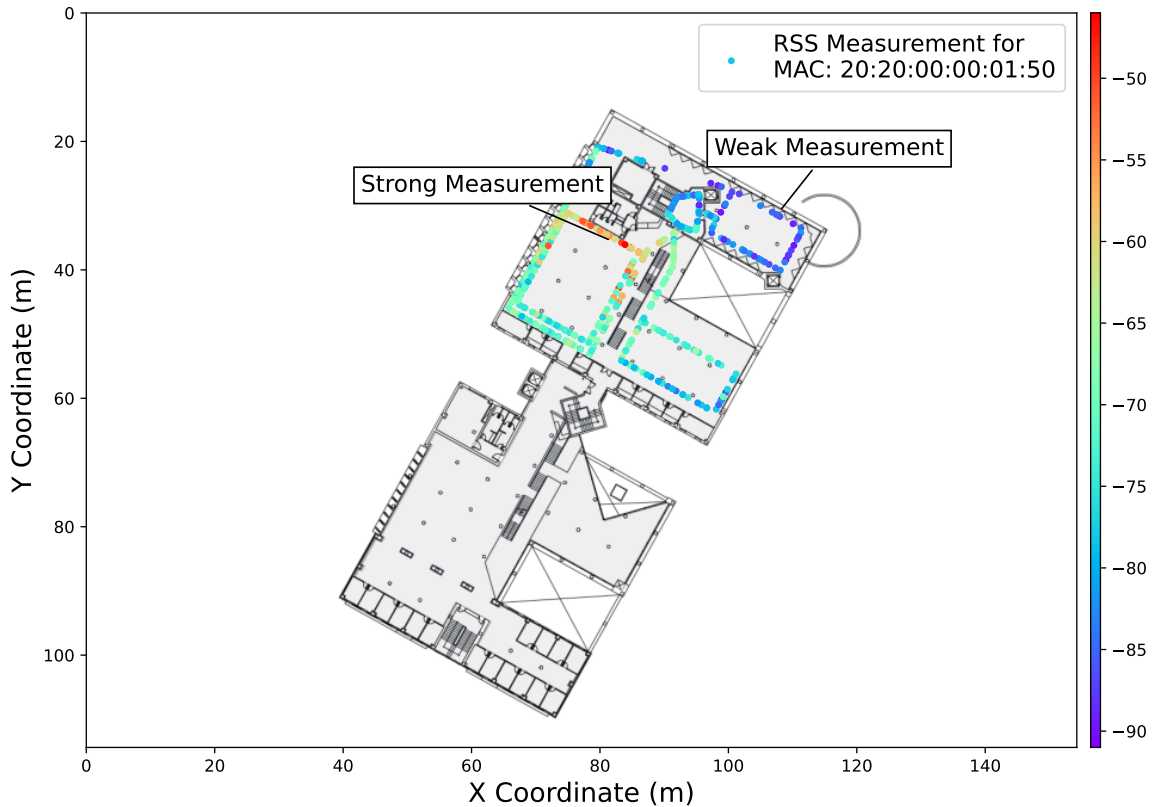


FIGURE 5.8: Interpolation of WiFi measurement position for MAC 20:20:00:00:01:50 at level 2.

5.2.3.2 Data Filtering

The main objective of this section is to evaluate the fidelity of wireless data generation using real data. This evaluation involves a comparison in terms of similarity of data values and similarity of data distribution. These two metrics are also employed in training WirelessDT with real data. Consequently, a statistical analysis of the data on WiFi measurements in the IPIN dataset is first conducted.

Floor Filter

Initially, it is necessary to filter the floor where the WiFi Access Point (AP) corresponding to the MAC address is located, aiding in determining the location of the WiFi AP. In the subsequent training of WirelessDT, estimating the location of the WiFi APs is required.

Figure 5.9 displays the distribution of signals corresponding to MAC address on different floors. It is believed that the floor with the stronger signal distribution and a higher number of acquisitions is where the MAC address corresponding to the AP is located. Based on this assumption, statistical analysis methods can be used to estimate the floor where the AP is located.

the floor of APs is estimated by identifying the floor that exhibits the highest frequency of top- n strongest RSS measurements. This can be represented mathematically as:

$$Floor(mac) = \arg \max_{floor \in floors} floorfrequency(floor, RSS_n) \quad (5.3)$$

Here, $Floor(mac)$ denotes the estimated floor corresponding to a given MAC address. RSS_n is the set of top- n strongest RSS measurements for the same MAC address. The floor where AP is situated is determined by finding the maximum floor frequency among these signal strengths, applying the $\arg \max$ operation.

Figure 5.10 demonstrates the correlation between the use of various n values in floor detection and the subsequent filtering of measurements based on these detection outcomes. The x-axis denotes various n values, while the y-axis signifies the fraction of filtered data relative to the original data, with a higher percentage indicating superior data preservation. The figure

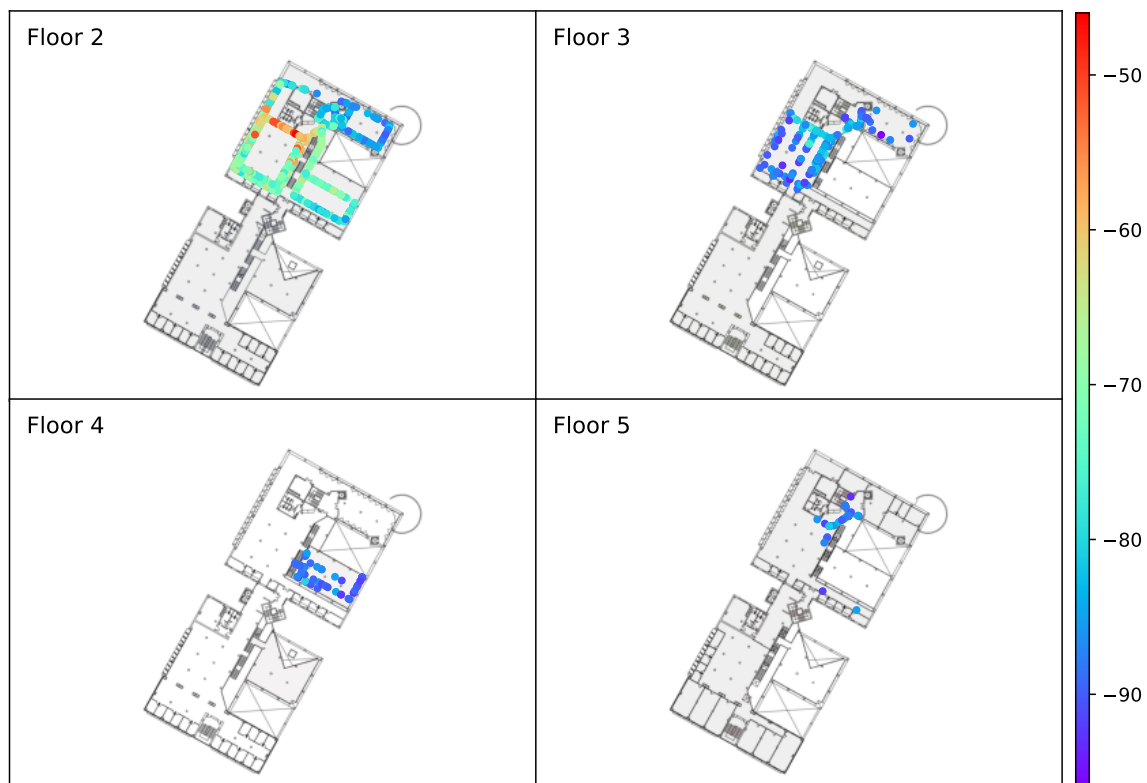


FIGURE 5.9: RSS distribution for MAC 20:20:00:00:01:50 at different floor.

reveals that varying n values exert a negligible influence on filtering results, maintaining data retention within a narrow band of 46% to 48%. In particular, as n exceeds 17, data retention experiences a slight downward trend with an increase in n . Thus, in subsequent experiments, this study adopt $n = 10$ as the optimal value for floor detection and filtering.

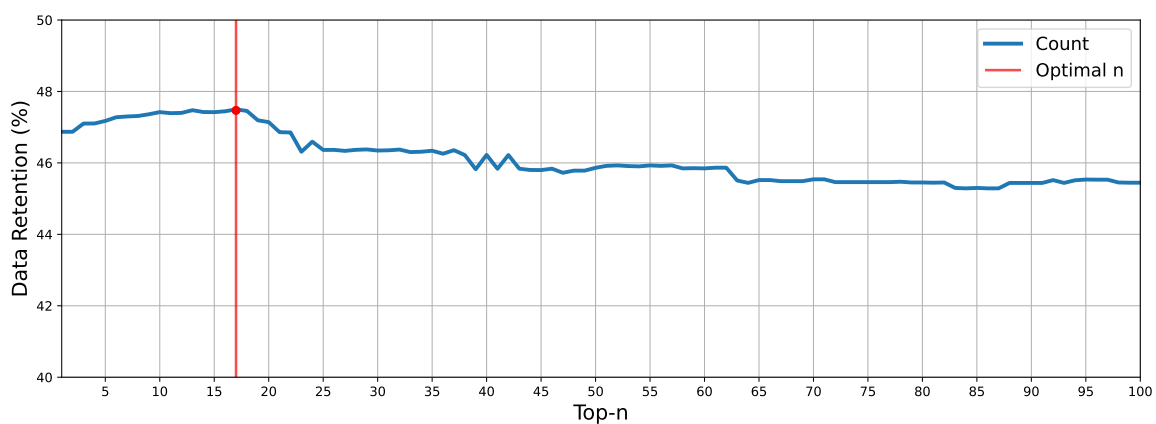


FIGURE 5.10: Search for optimal n value for floor detection.

Quantity Filter

In subsequent experiments, training WirelessDT using robust real-world data is crucial, enabling performance optimization by refining its parameters. A key criterion for the training dataset is its size, with a substantial amount of data necessary to effectively capture the variations and nuances of the task. During the training process, a primary objective is the accurate estimation of the location of AP associated with a specific MAC address. For this purpose, the model is trained exclusively on measurements corresponding to a single MAC address. Measurements with few observations can compromise the model's efficacy, leading to their exclusion based on a predefined quantity threshold. Figure 5.11 illustrates the number of measurements for each MAC address that exceeds a quantity of 100. Subsequent experiments will focus solely on data surpassing this quantity threshold. This filtering process helps maintain the data's quality, ensuring the model's reliability and precision.

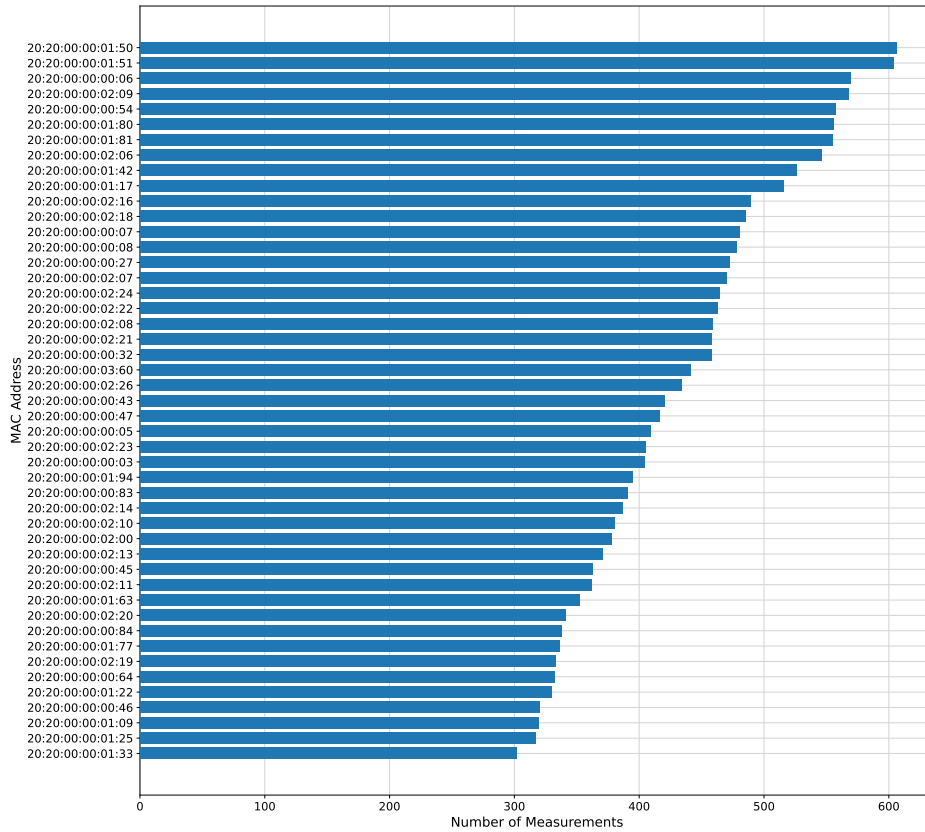


FIGURE 5.11: The number of measurement for each MAC (with quantity greater than 300).

5.2.3.3 Data Augmentation

In practical scenarios, a single **AP** may be equipped with multiple **Network Interface Cards (NICs)**, each corresponding to a unique physical **MAC address**. Consequently, within the **RSS** measurement data, measurements derived from distinct **NICs** of the same **AP** may be encountered. Taking this into account, an effective strategy to augment training data involves overlaying measurements from different **MAC addresss** that belong to the same **AP**. Since these **MAC addresss** essentially represent the same **AP** location, their combined measurements can provide a more comprehensive observation, thus enriching the dataset and improving training effectiveness.

Figure 5.12 displays the diverse distribution of **RSS** within the assembled dataset, featuring measurements from four distinct **MAC addresss** situated on separate floors. The figure highlights the variation in distribution patterns among different **MAC address**.

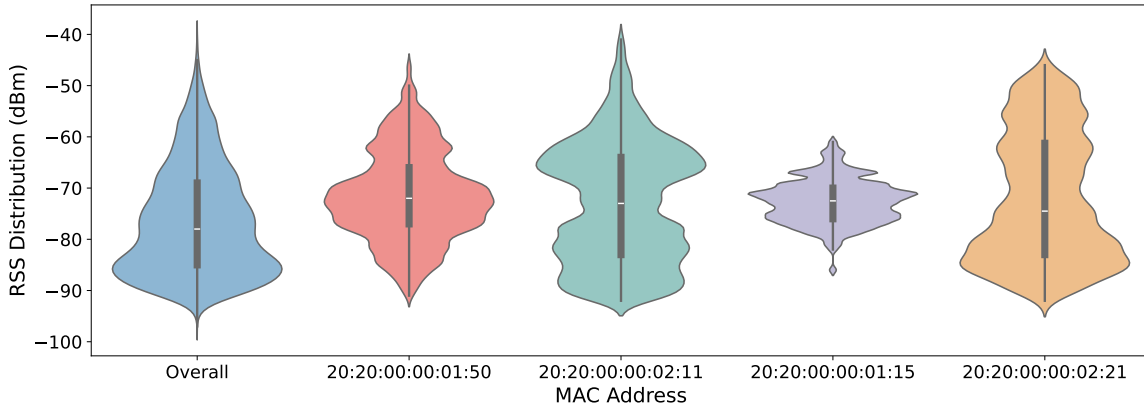


FIGURE 5.12: RSS probability distribution over different MAC address.

To determine **MAC address** addresses related to the same **AP**, **JS Divergence** is utilized as a metric to compare signal strength measurement distributions across all **MAC addresss**.

Fig. 5.13 presents a cross-comparison of the RSS distribution using **JS Divergence**. Blue indicates a similarity in distribution patterns, whereas red signifies a significant divergence. As depicted, certain distributions share similar patterns, suggesting that they might be related to the same **AP** on the same floor.

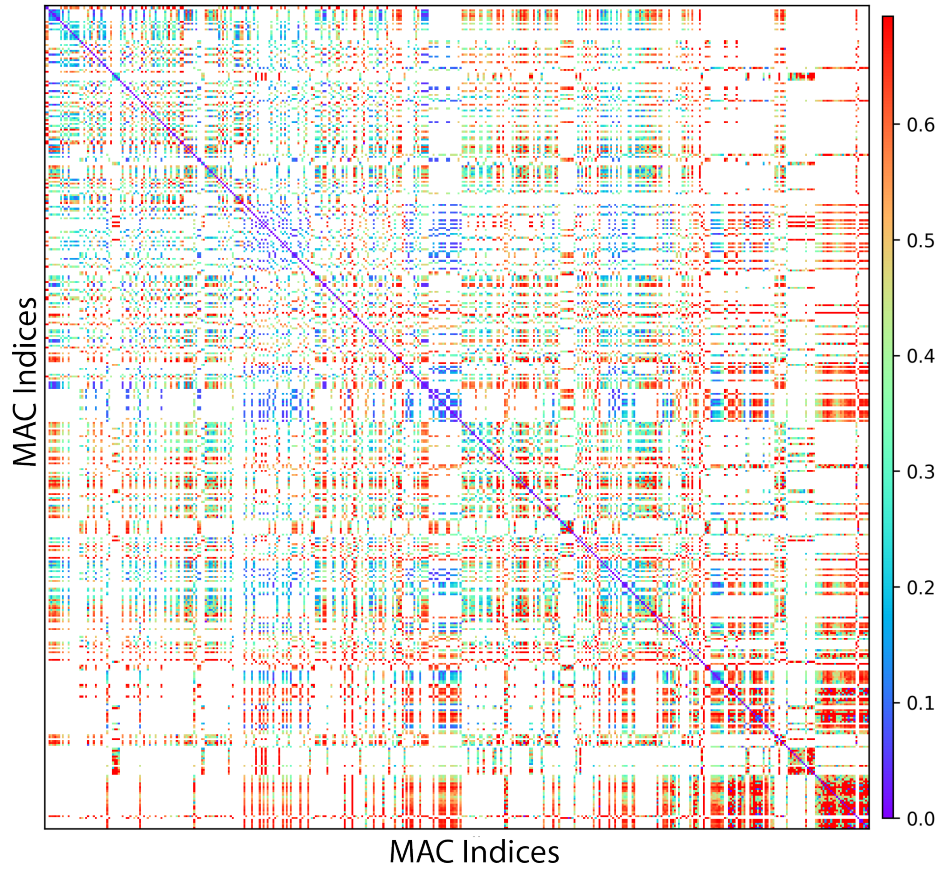


FIGURE 5.13: Cross comparison of RSS measurement distribution. White area indicate the MAC addresses is not belong to access point from same floor.

Fig. 5.14 shows that three pairs of MAC address exhibit low JS Divergence upon cross-comparison, indicating strong similarities in their distributions.

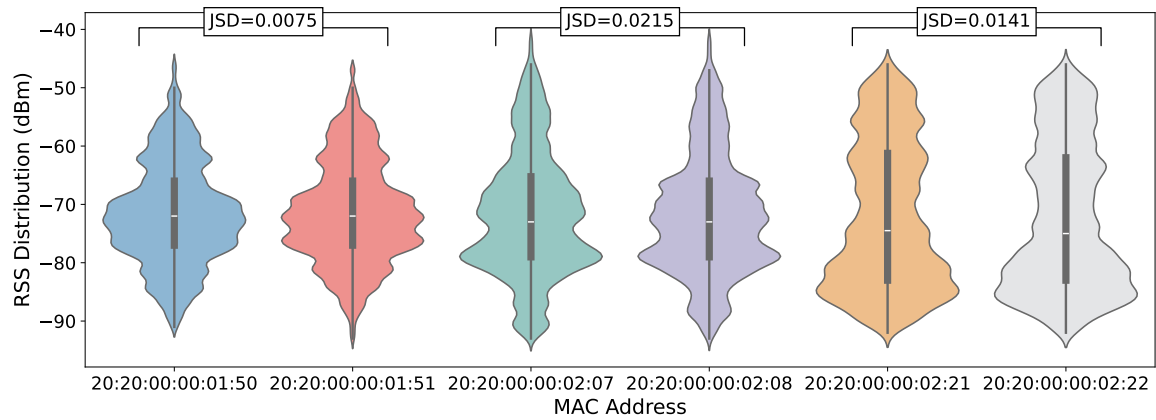


FIGURE 5.14: Search similar distribution with JS divergence.

5.2.4 Training WirelessDT

5.2.4.1 Training Process

The aim of training WirelessDT is to enhance the simulation quality by automatically adjusting the model parameters through learning from real data. This section demonstrates the training of WirelessDT using WiFi wireless signals, in conjunction with the IPIN data set. The training focuses on WirelessDT with location information x and RSS measurements y , enabling it to generate RSS measurements $\hat{y}$ corresponding to any specified location. The data used were processed by interpolation, filtering, and augmentation using the feature engineering methods outlined in Section 5.2.3.

As outlined in the previous section 4.5.2, WirelessDT is perceived as a parameterized generative model $\mathcal{G}_\psi = \mathcal{G}_{\theta, \phi}$. Here, the non-trainable parameters ϕ represent the building model, constructed based on known indoor maps and literature. The trainable parameters θ correspond to the unknown AP position and environmental factors. A penetration factor ρ and a reflection factor γ are defined as environmental factors for training. The penetration factor ρ modulates the attenuation rate of wireless signals penetrating geometric bodies, while the reflection factor γ controls the attenuation rate of reflected wireless signals. For simplicity, global penetration and reflection factors are defined for training, although different parameters can be defined for different materials for more granular training.

To optimize the model parameters, adherence to the methodology presented in Section 4.5.3 is maintained. The performance of model generation is assessed by calculating the ED between the real and generated data, and the JS Divergence of the data distribution as the objective function, expressed as follows:

$$\mathcal{L} = \frac{1}{M} \sum_{i=1}^M |\hat{y}^{(i)} - y^{(i)}| + \beta D_{JS}(p(\hat{y})||q(y)) \quad (5.4)$$

where M is the total number of data points to be compared. $\hat{y}^{(i)}$ is the i^{th} generated data from the WirelessDT generative model $\mathcal{G}(x)$, and $y^{(i)}$ is the corresponding true data from the real dataset. Specifically, y is the measured value of RSS at a given position.

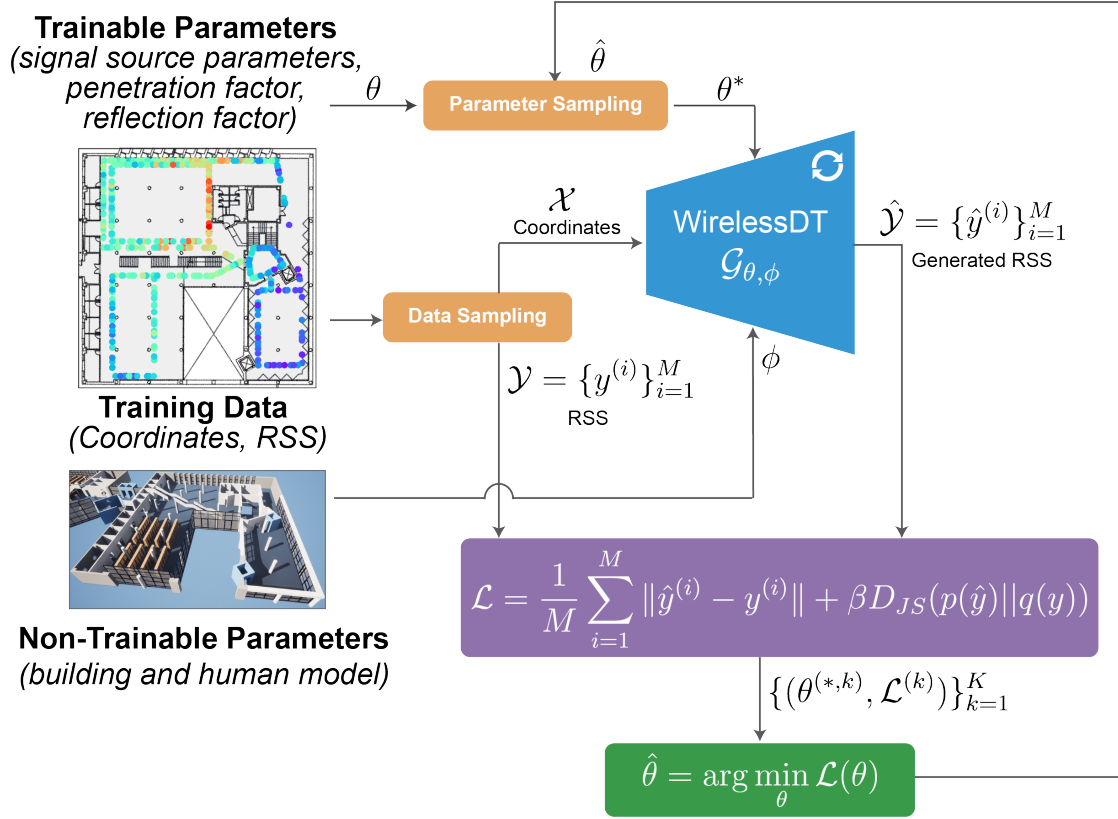


FIGURE 5.15: The overview of training process.

Fig. 5.15 illustrates the WirelessDT training process, where the inputs for model generation are categorized into three main parts: training data, trainable parameters, and non-trainable parameters. Initially, the model accepts the building and human models as non-trainable parameters ϕ . The trainable parameters θ are then initialized. Based on these initialized parameters and boundary conditions, θ is sampled to obtain θ^* .

For example, with the knowledge that an AP is on a certain floor and the dimensions of that floor, uniform sampling of conditions on the floor is conducted to derive the actual input parameter θ^* . The dataset is then sampled, numerous pairs of features and labels extracted, and segregated into a set of features $\mathcal{X}$ and a set of labels $\mathcal{Y}$, where the features are the coordinates of the measurements, and the labels are the corresponding RSS measurements.

For a single measurement coordinate $x^{(i)}$, WirelessDT generates the corresponding measurement result $\hat{y}^{(i)}$, with $\hat{\mathcal{Y}}$ being the set of these results. Using $\mathcal{Y}$ and $\hat{\mathcal{Y}}$, the probability

density distributions of **RSS** measurements with respect to the signal strength $p(\hat{y})$ and $q(y)$ are determined. Consequently, the model's loss $\mathcal{L}$ is computed by calculating **ED** and **JS Divergence**. For each sampling parameter $\theta^{(*,k)}$, the corresponding loss $\mathcal{L}^{(k)}$ is calculated sequentially. By minimizing $\mathcal{L}^{(k)}$, the best matching result $\hat{\theta}$ is obtained. Based on this, local sampling based on $\hat{\theta}$ is performed to train the parameters with finer granularity.

5.2.4.2 Parameter Sampling

Specifically, the training parameters θ , including the 3D coordinates of **AP** (a_x, a_y, a_z), as well as the global penetration factor ρ and the global reflection factor γ , are required.

The sampling strategy can be adjusted according to the training phase. Based on the analysis of the raw data, the floor where **AP** is located can be determined. With the knowledge of the floor's height and the spatial distribution of the signal measurement data, the possible range of the coordinates can be roughly determined. This means that the boundary conditions of the **AP** can be obtained through the preconditions, as shown in Fig. 5.16.

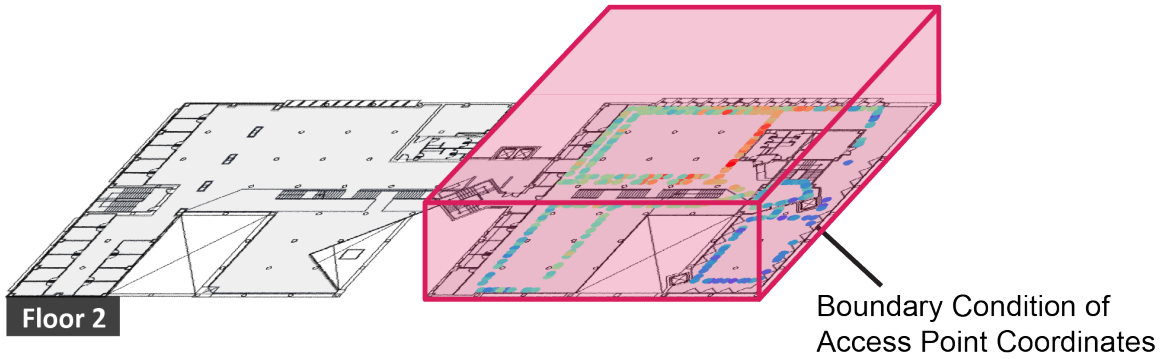


FIGURE 5.16: Determining access point coordinates boundary conditions.

Another approach involves using the least squares method for an initial estimate of the location of **AP**, followed by sampling it using a Gaussian method for greater efficiency. To better illustrate the relationship between parameter variations and fitting results, the method of uniform sampling under the given boundary conditions will be employed for subsequent experiments.

Additionally, with the known range of transmit power (-30 dBm to 30 dBm), and the range of the penetration and reflection factors $\rho, \gamma \in [0, 1]$, uniform sampling of these six dimensions $(a_x, a_y, a_z, p_{Tx}, \rho, \gamma)$ as trainable model parameters is feasible. Due to the extensive parameter space and depending on the correlation between the parameters, the training is divided into two steps:

- Step 1: Given the rough (a_x, a_y, a_z) values, sample (p_{Tx}, ρ, γ) for training and find the corresponding losses.
- Step 2: Given (p_{Tx}, ρ, γ) that minimizes the loss in Step 1, the training model can be trained with (p_{Tx}, ρ, γ) for the lowest loss in 1). sample (a_x, a_y, a_z) and compute the loss, and solve for the optimal value (a_x, a_y, a_z) .

The left-hand side of Figure 5.17 displays the **ED** loss, **JS Divergence** loss, and their combination, respectively, as functions of penetration, reflection, and transmit power. The red dots indicate the coordinates that minimize the value of the **JS Divergence** loss function. The right side illustrates the comparison of WirelessDT generated data with real data, using the optimal value as a parameter. By combining **ED** loss and **JS Divergence**, improved performance is achieved, as demonstrated in Figure 5.17(c).

The left-hand side of Figure Fig. 5.18 shows the **ED** loss, **JS Divergence** loss, and their combination, respectively, as a function of coordinates. On looking at the figure, it can be seen that the loss is not sensitive to the change in the z-axis, that is, the height parameter of **AP** has little effect on the results. The right side shows the results of WirelessDT generated data compared with the real data when using the optimal value as a parameter. The **ED** loss focuses on the difference in absolute value between data points, which makes the data generated by the optimal parameter based on euc**ED** loss more similar in spatial distribution, and its mean and variance are closer to the original value. **JS Divergence** focuses on the similarity between the overall distribution of the generated data and that of the original data, even though they are more similar in spatial distribution. **JS Divergence** focuses on the similarity between the overall distribution of the generated data and the original data, although their spatial distributions are more different, but in the overall pattern of the probability distribution it is closer. Figure 5.18(c) demonstrates the combination of the previous two, which makes the

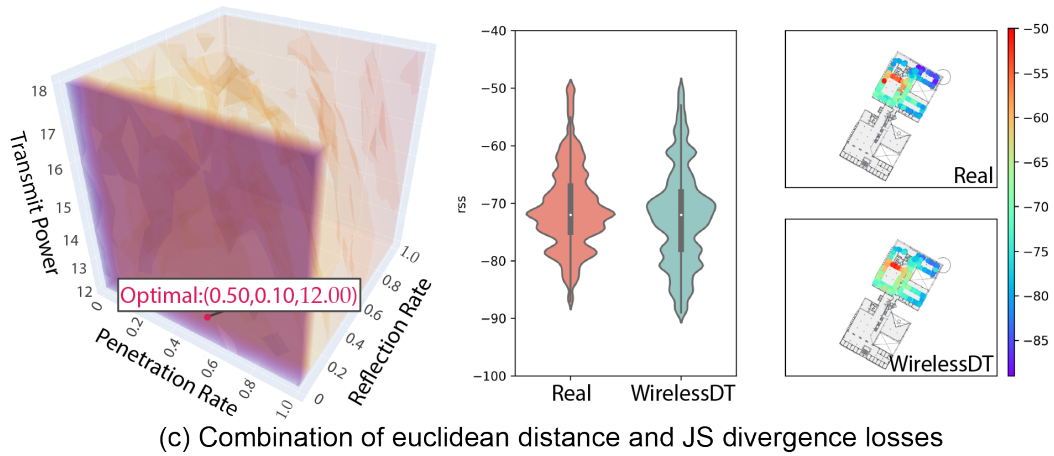
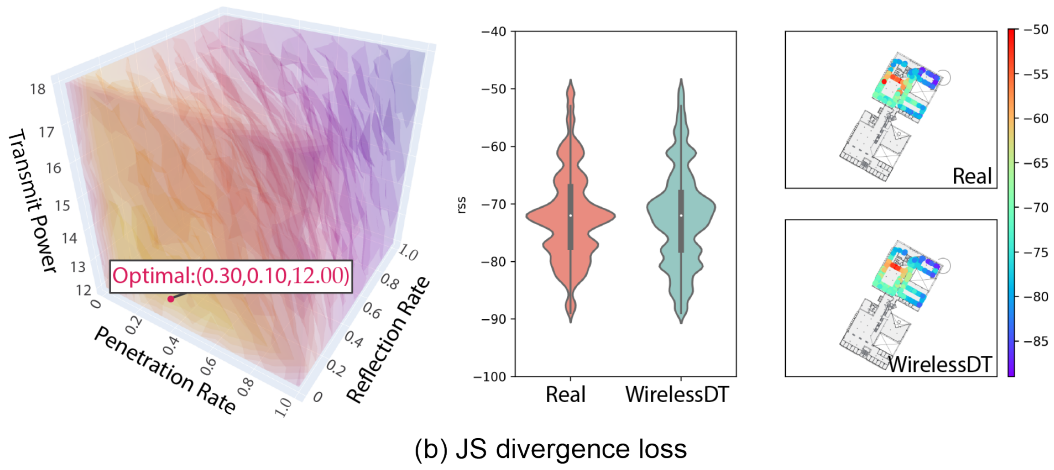
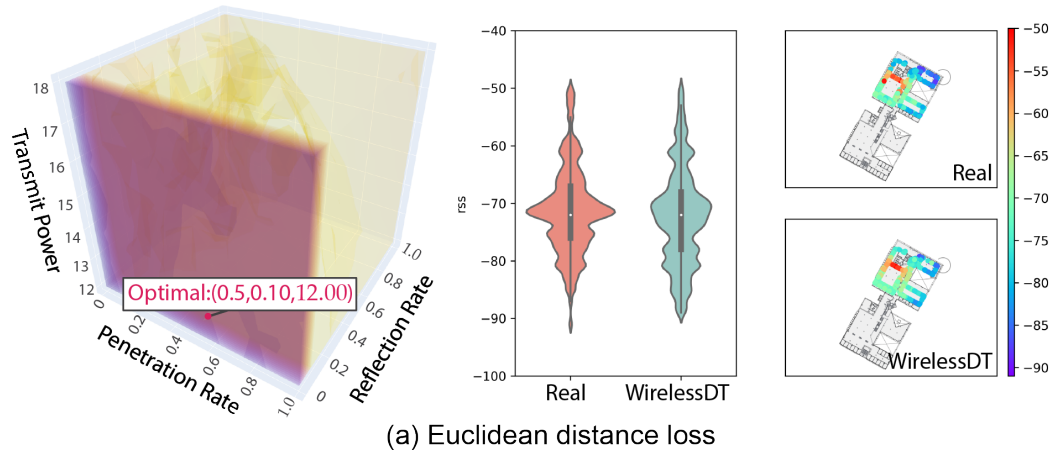


FIGURE 5.17: Loss space visualization cross penetration factor, reflection factor and transmit power.

data generated in the spatial distribution and the overall distribution of the performance of the data good.

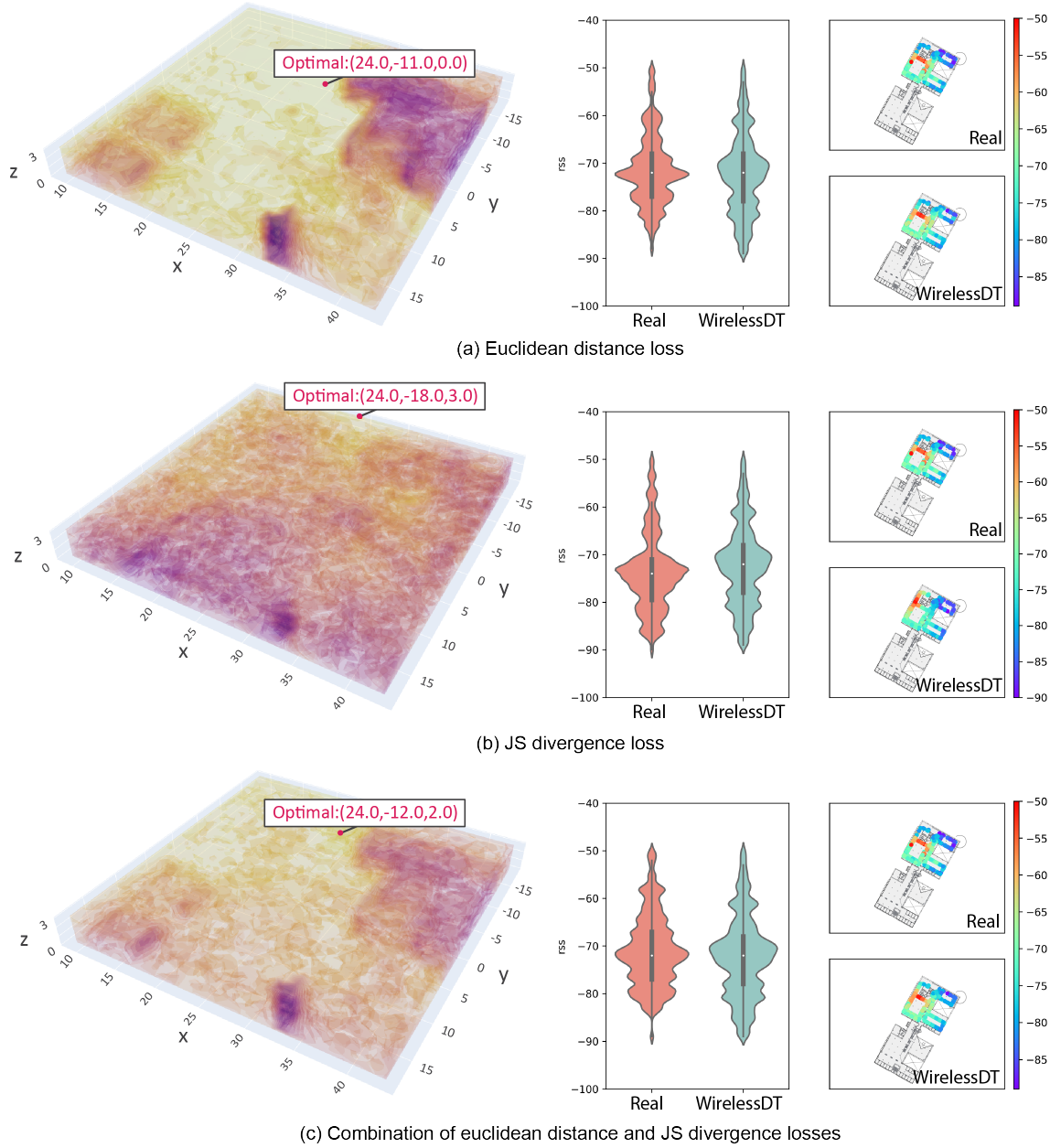


FIGURE 5.18: Loss space visualization cross signal source position.

Finally, through iteration and narrowing down of the parameter sample boundaries, the final suitable parameter combinations can be obtained.

5.2.5 Experimental Results

5.2.5.1 Benchmark

The classic path loss model (Neskovic *et al.*, 2000) for indoor radio propagation modeling was selected as the baseline for comparison. The path loss model is defined by the following equation:

$$\mathcal{L}_{\text{path}}(d) = \mathcal{L}_{\text{path}}(d_0) + 10n \log\left(\frac{d}{d_0}\right) + X_\sigma \quad (5.5)$$

Here, $\mathcal{L}_{\text{path}}(d)$ represents the path loss in dB at distance d , $\mathcal{L}_{\text{path}}(d_0)$ denotes the known path loss at the reference distance, n signifies the path loss exponent dependent on the signal propagation environment, and X_σ accounts for noise terms. Given that the transmitter position, the path loss exponent, and the path loss at the reference distance are unknown, the original path loss model is modified as follows:

$$\mathcal{L}_{\text{path}}(p_{\text{Tx}}, p_{\text{Rx}}, n, \mathcal{L}_{\text{path}_0}) = \mathcal{L}_{\text{path}_0} + 10n \log(\|p_{\text{Tx}} - p_{\text{Rx}}\|_2) + X_\sigma \quad (5.6)$$

In this equation, $\mathcal{L}_{\text{path}}(p_{\text{Tx}}, p_{\text{Rx}}, n, \mathcal{L}_{\text{path}_0})$ represents the path loss determined by the transmitter position p_{Tx} , receiver position p_{Rx} , path loss exponent n , and path loss at a 1 meter distance. To apply the baseline model, least-squares regression is used with the preprocessed dataset, which contains pseudo ground truth positions of the received locations and RSS measurements. This regression fits the parameters of the baseline model, including the transmitter position p_{Tx} , the path loss exponent n , and the path loss at 1 meter. With the fitted model parameters $\hat{p}_{\text{Tx}}$, $\hat{n}$, and $\hat{\mathcal{L}}_{\text{PL}_0}$, the baseline model can be presented as:

$$\mathcal{L}_{\text{PL}}(p_{\text{Rx}}) = \hat{\mathcal{L}}_{\text{PL}_0} + 10\hat{n} \log(\|\hat{p}_{\text{Tx}} - p_{\text{Rx}}\|_2) + X_\sigma \quad (5.7)$$

Given the receiver position p_{Rx} , the baseline model can estimate the RSS in dBm.

5.2.5.2 Metric

Five metrics, namely KL Divergence, JS Divergence, MAE, MSE, and the R^2 score is used at four levels (2 to 5) for the evaluation. From a distribution point of view, KL Divergence and JS Divergence are employed to measure the similarity between the generated and real data.

KL Divergence is an asymmetric measure that quantifies the difference between an estimated probability distribution P and the true distribution Q , defined as:

$$D_{\text{KL}}(P||Q) = \sum P(x) \log \left(\frac{P(x)}{Q(x)} \right) \quad (5.8)$$

. **JS Divergence** is a finite symmetric divergence derived from **KL Divergence** as follow:

$$D_{\text{JS}}(P||Q) = \frac{1}{2}D_{\text{KL}}(P||M) + \frac{1}{2}D_{\text{KL}}(Q||M) \quad (5.9)$$

, where $M = \frac{1}{2}(P + Q)$. It is believed that realistically generated data should exhibit a distribution similar to real-world measurements, resulting in small **KL Divergence** and **JS Divergence** values. From a numerical perspective, **MSE** and **Mean Absolute Error (MAE)** are used to evaluate the differences between generated and real-world measurements. Regarding the model, the **R^2 score**, also known as the coefficient of determination, is employed to assess its performance, defined as:

$$R^2 = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y})^2} \quad (5.10)$$

, where y_i represents the observed data, $\bar{y}$ is the mean of the observed data, and f_i is the predicted value from the model. An R^2 value of 1 indicates that the model perfectly captures the variance of the data, while an R^2 value less than 0 implies a worse model performance than simply using the mean value for prediction.

5.2.5.3 Discussion

An evaluation was conducted to assess the fidelity of WirelessDT (WDT) by comparing the generated emulation data with a public dataset. For a fair and accurate comparison, data was generated at the same positions as the measurements in the dataset. Benchmark data for comparison was also generated using the baseline model (BL). The final results, as shown in Table 5.1, indicate that from the perspective of data distribution, WirelessDT significantly improved compared to the baseline, especially in terms of **KL Divergence**. Numerically, WirelessDT surpassed the baseline model at all levels. Moreover, WirelessDT achieved better model goodness, reflected in higher R^2 scores across all experiments. Figure 5.19 displays the real dataset, data generated by the benchmark and WirelessDT, their differences, and the

probability distribution of the real and generated data. These results demonstrate that the generated data closely resemble real measurements in terms of position and distribution. Even at level 5, which had the highest [JS Divergence](#) value of 0.1281, the fidelity and similarity of the generated data were remarkable. This experiment confirms that the WiTracing engine in WirelessDT effectively simulates signal propagation behavior, accounting for both stationary and dynamic features of the environment, and generates highly reliable RSS measurements after proper calibration.

TABLE 5.1: The experimental result of fidelity evaluation for WirelessDT(WDT) and Baseline (BL)

	Level 2		Level 3		Level 4		Level 5	
	WDT	BL	WDT	BL	WDT	BL	WDT	BL
KL Divergence ↓	0.4847	0.8218	0.8610	1.9873	1.5450	5.0889	1.4702	3.3884
JS Divergence ↓	0.0643	0.0652	0.0974	0.1224	0.0655	0.1486	0.1281	0.1442
MAE ↓	3.8730	4.7302	7.5758	8.0202	3.1538	4.1731	5.7431	7.0092
MSE ↓	24.7090	40.1058	94.7879	144.8081	16.3462	32.7115	52.4771	86.8257
R^2 ↑	0.6550	0.4401	0.2784	-0.1024	-0.5254	-2.0526	0.6035	0.3439

5.3 Practicality Evaluation

To demonstrate the practicability of WirelessDT, its application in wireless indoor localization is showcased. A mobile localization application on iOS was developed for this purpose. The application utilizes two classical indoor localization algorithms: [Kalman Filter \(KF\)](#)-based iteration and [Recurrent Neural Network \(RNN\)](#). Concurrently, a 1:1 virtual representation of the physical scene for WirelessDT calibration and a large-scale virtual environment for testing the mobile localization application were constructed. Additionally, a software interface was established to connect the device with WirelessDT. Real-time generated data is transmitted to the mobile device, enabling the activation of positioning algorithms. The positioning results obtained in real time are compared with the ground-truth data within a flexible virtual representation, allowing for a comprehensive evaluation of the application's performance across various scenarios, both quantitatively and qualitatively. The ease of use, flexibility, and low cost of ownership of WirelessDT provide new options for testing wireless applications.

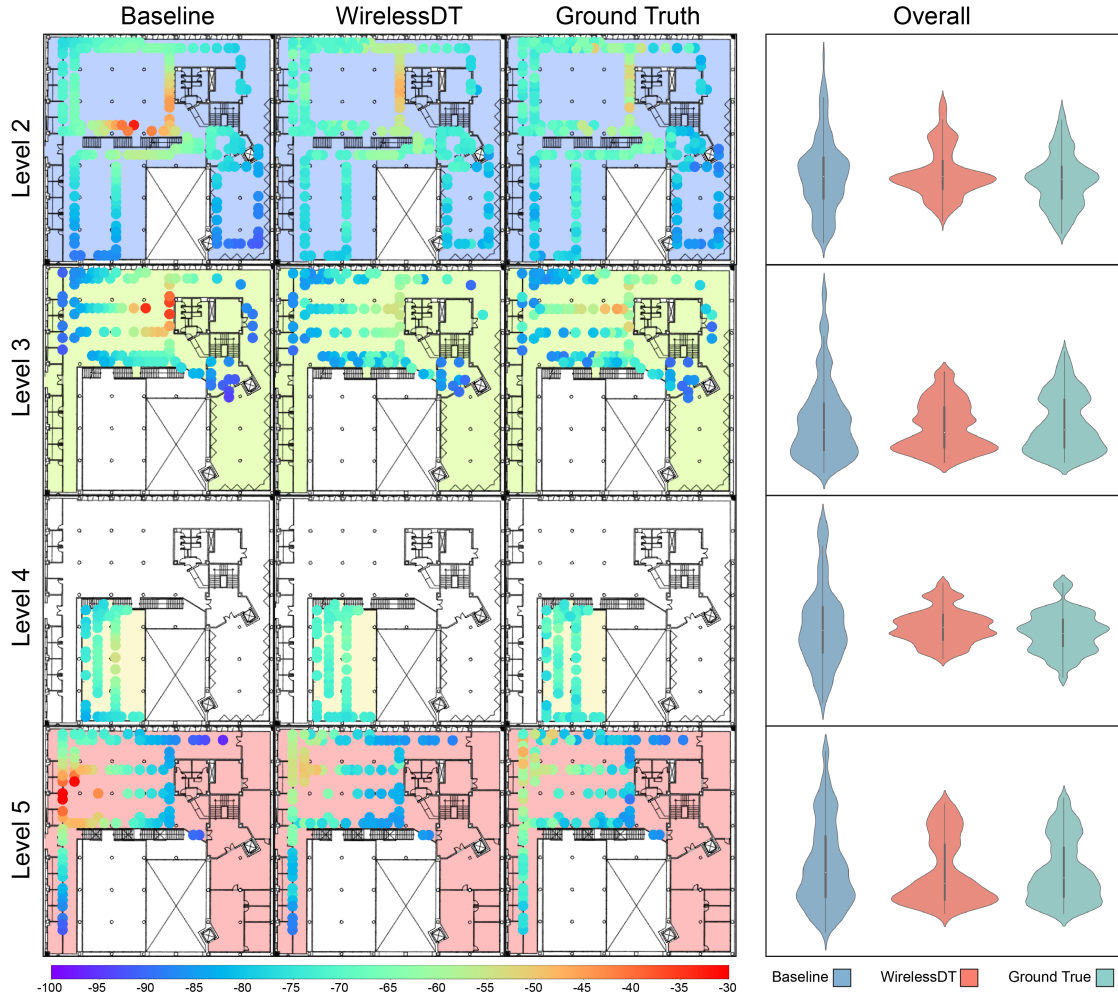


FIGURE 5.19: Comparison of received signal strength distribution on different level.

5.3.1 Environmental Modeling

In practical applications, environmental parameters can be learned by modeling a small-scale environment and calibrating the virtual model during the virtual reality synchronization process. These parameters are then applied to a larger-scale model for actual testing. To facilitate this, a small-scale real environment is modeled, and a fictionalized, realistic large-scale office scene is created.

Figure 5.20 shows an online calibration scenario in a small-scale environment. The subject part of the figure constitutes the virtual representation that has been constructed. The upper right part is the real physical world. The wireless signals **Tx** and **Rx** in the virtual environment

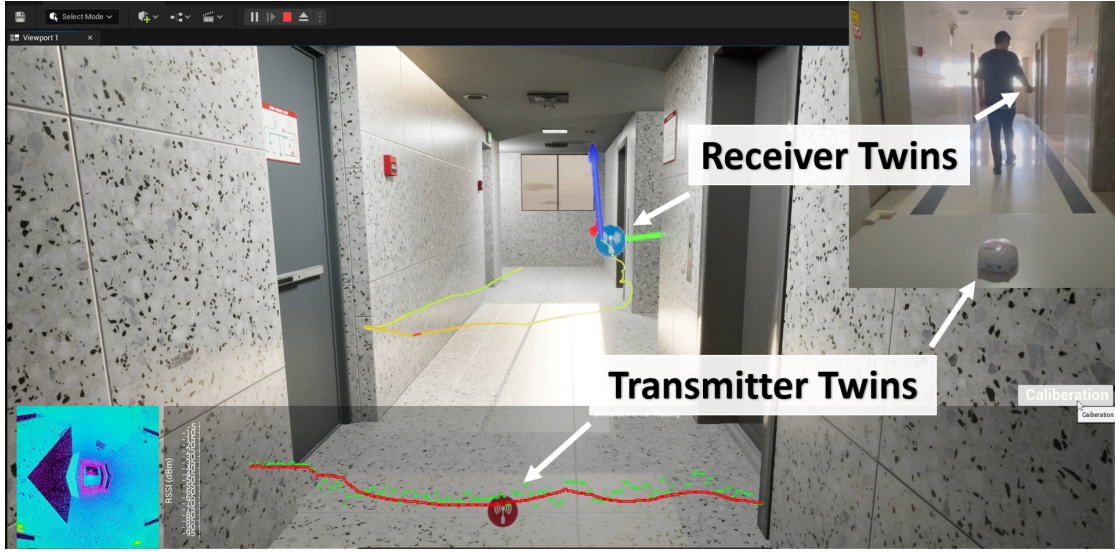


FIGURE 5.20: Online calibration in small scale corridor environment.

have corresponding physical entities. During the calibration process, the researcher moves around the scene holding the **Rx**. The physical twin (the carrier of **Rx**) acquires its relative position and orientation in the scene through built-in sensors and synchronizes the position information to the virtual environment through an internal communication interface. The position of **Rx** in the virtual environment remains the same as in the real scene. During the data synchronization process, the wireless measurement data is also synchronized to the virtual environment. WirelessDT simulates wireless signals in real time during the synchronization process and self-calibrates by evaluating the difference between the simulation results and the real-time measurements. The line graph at the bottom of the figure shows the difference between the real data and the simulation results after the calibration process. The trajectory lines in the scene indicate the degree of fit at each location during the calibration process, with the size of the error indicated by the red to green colors, where red indicates a larger error, and green indicates a smaller error.

In Figure 5.21, the virtual modeling of a large-scale office scene is depicted. The red icon represents the wireless signal transmitter (**Tx**), while the blue icon symbolizes the wireless signal receiver (**Rx**). The **Rx** corresponds to a cell phone, and the data collected by it will be synchronized in real time with a genuine cell phone application during the testing process. To recreate a lifelike environment, 80 **AI**-controlled characters have been integrated into the

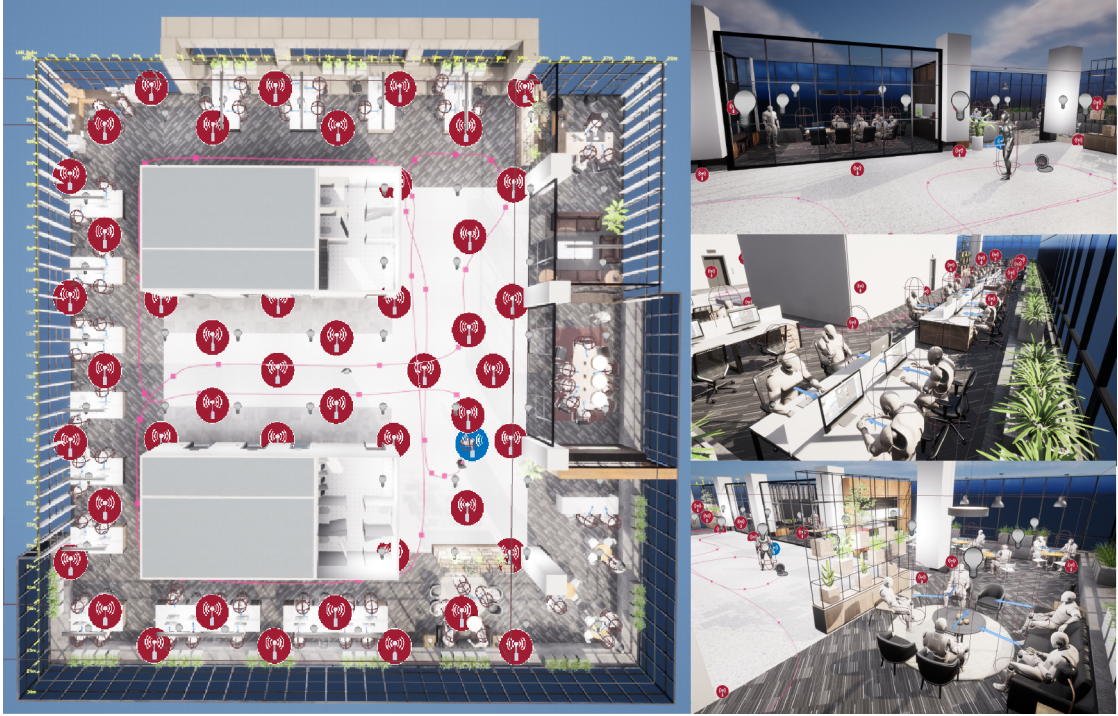


FIGURE 5.21: Testing environment of large-scale office scene.

scene, each operating under distinct **AI** logic to imitate human actions such as conversation, movement, and work. Notably, the wireless signal **Rx** (blue icon) is attached to one of these **AIs**, replicating a pedestrian holding a mobile device. Throughout the simulation, the signal received by the mobile device dynamically changes in response to its position shifts. By combining dynamic scenario simulation with pre-established parameters, authentic real-time data can be generated for the evaluation and testing of mobile wireless applications.

5.3.2 Example Application

To showcase the practicality of the WirelessDT platform to assist in the life cycle of wireless software applications, a mobile pedestrian positioning application is developed and incorporated two well-known localization algorithms: the **KF-based trilateration** and the **Deep Recurrent Neural Network (DRNN)**.

- The **KF**-based trilateration algorithm is a commonly used technique that uses **RSS** measurements to estimate distances based on the path loss equation. It employs

distance measurements from at least three signal sources with known positions to estimate the location of the mobile device in 2D or 3D space. To account for uncertainties and improve accuracy over time, **KF** is applied to refine the estimated position.

- On the other hand, the **DRNN** model is a popular deep learning-based algorithm used for pedestrian localization. It consists of multiple recurrent layers, enabling it to capture complex dependencies in sequential data. The **DRNN** process sequences of **RSS** measurements to predict the position of the mobile device. Similarly to the **KF**-based trilateration algorithm, **DRNN** considers the temporal dependencies in the device's trajectory, resulting in more stable predictions.

To facilitate the visualization and comparison of results, a **Graphical User Interface (GUI)** is implemented within the mobile application. This GUI enables us to visualize the localization performance in real time. Additionally, the mobile pedestrian positioning application integrates the WirelessDT software interface, allowing it to receive live wireless emulation data for real-time evaluation.

5.3.3 Results and Discussion

A real-time performance assessment of the wireless positioning mobile application is conducted utilizing WirelessDT. The positioning result is shown in Figure 5.22. A comparison between the ground truth trajectory and the estimated trajectory revealed that the **DRNN** algorithm provided more accurate and stable positioning results compared to the **KF**-based trilateration algorithm. Analyzing the probability distributions of position error, the **DRNN** algorithm exhibited a smaller mean and variance of position error compared to **KF**-based trilateration. The **DRNN** algorithm consistently maintained an error range within three meters, while **KF**-based trilateration achieved an error range within six meters. The cumulative distribution of the position error supported these findings, with more than 75% of the position results having an error of less than two meters for the **DRNN** algorithm, compared to fewer than 25% for **KF**-based trilateration. These results indicate that the **DRNN** algorithm is a more suitable choice for wireless pedestrian positioning scenarios. WirelessDT allows for

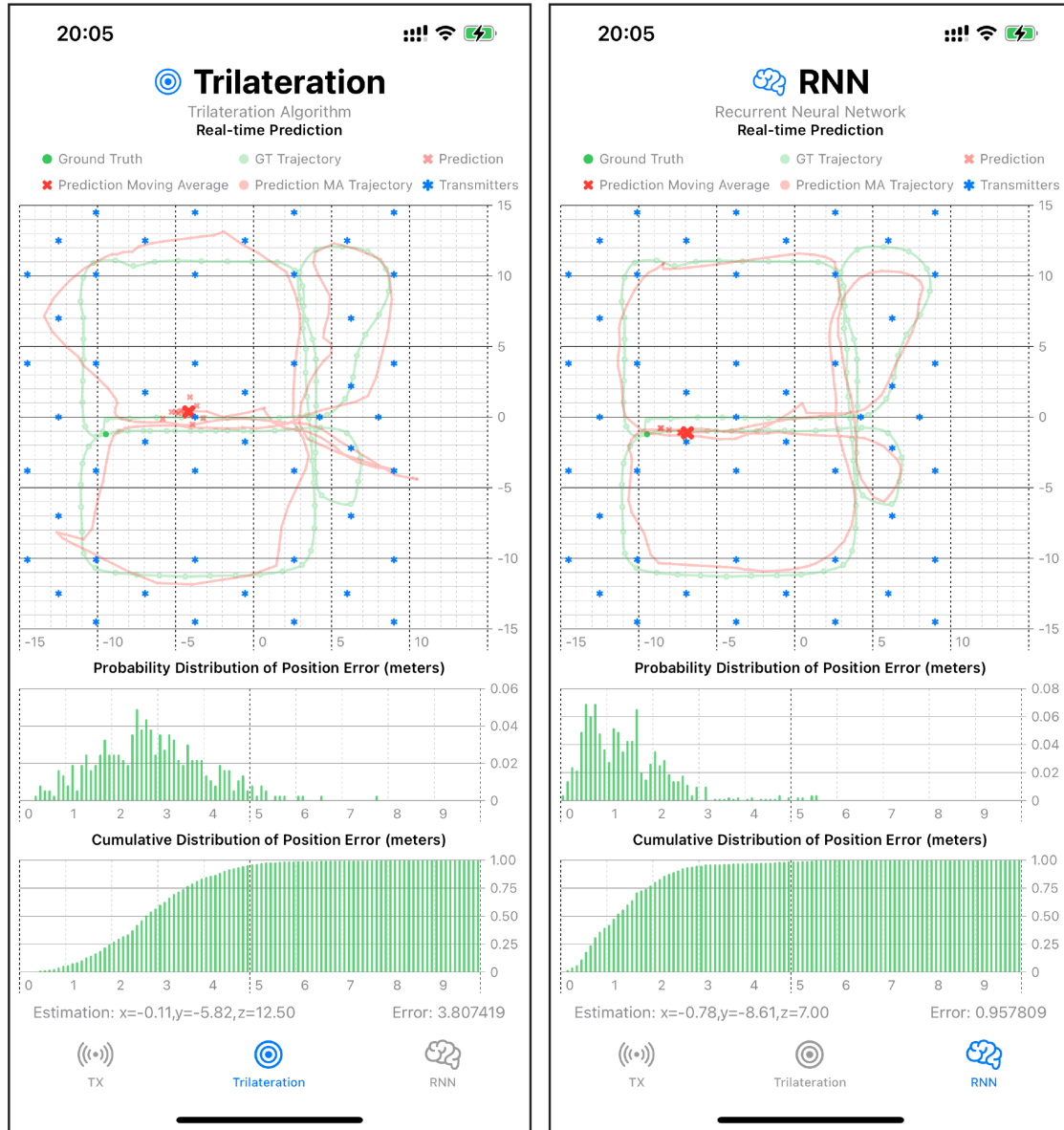


FIGURE 5.22: The evaluation of wireless positioning application.

easy adjustment of the virtual scene to evaluate wireless software applications in various scenarios, minimizing cost and effort. Moreover, real-time evaluation provides the opportunity to examine the performance in operation of wireless software applications and support SIL testing. These advantages are particularly beneficial for tasks such as positioning indoors that require extensive labor for a thorough evaluation.

5.4 Scalability Evaluation

Scalability is an important aspect in terms of large-scale real-time applications. WirelessDT is compared with two widely used simulation tools, namely Matlab and Carla. Matlab offers a built-in library for radio frequency analysis that uses **RT**, while Carla employs **CPU RT** in Unreal Engine to simulate **RADAR** and **LiDAR** sensors in autonomous driving systems.

Although Matlab, Carle, and WirelessDT all adopt the light pursuit technique to realize the wireless signal propagation simulation, but their ways of using application scenarios and parameter settings are different, so it is difficult to compare them under the same scenario and environment settings. The performance of **RT** greatly determines the scalability of the simulation system, so the performance of **RT** in various aspects is compared, including whether it supports dynamic scenes, whether it can simulate in real time, whether it supports GPU **RT**, and the number of **RT** runs per second. A detailed comparison is presented in Table 5.2.

TABLE 5.2: The experimental result of scalability evaluation

	Matlab	Carla	WirelessDT
Dynamic Scene	×	✓	✓
Real-Time	×	✓	✓
GPU Ray Tracing	×	×	✓
Ray Tracing Per Second	10^2	10^6	10^8

Since there is a large difference in the order of magnitude of **RT** that can be performed per second between the three, they were countered in different ways.

- For MATLAB, the time required for a single simulation between transceivers and the quantity of ray traces are computed.
- For Carla, experimentation was conducted on the **LiDAR** sensor by adjusting the "points per second" attribute. The findings revealed a significant decrease in the Carla simulator's frame rate per second when simulating light more than 10^6 times per second, ultimately leading to a deadlock before reaching 10^7 times per second.

- For WirelessDT, the simulation speed was assessed by determining the number of **RT** per frame time, denoted as **Frames Per Second (FPS)**.

Although Matlab provides extensive libraries for wireless signal analysis, its scalability is limited by the performance of **RT** and its narrow scope, limiting real-time analysis capabilities. On the other hand, Carla serves as a simulation platform for autonomous driving and supports real-time **RT** in dynamic scenes for sensor simulation. However, due to the constrained performance of **CPU RT**, Carla struggles to handle large-scale deployments of wireless sensor systems beyond its intended scope. In contrast, WirelessDT surpasses Matlab and Carla in terms of **RT**-based sensor simulation, as it is purposefully designed for wireless signal simulation.

Fig. 5.23 illustrates how the frame rate varies with the number of pairs of wireless transceivers simulated per frame. The different line segments represent the changes in relationships under different sensor resolution conditions. When the sensor resolution is large (i.e., the resolution is greater than 64×64), the frame rate decreases more rapidly as the number of pairs of analog per frame increases. However, when the sensor resolution is low (i.e., less than 64×64), the resolution size has less impact on the frame rate. When only a single pair of wireless transceivers is simulated per frame, the frame rate can exceed 55 **FPS**, which is perfectly suited to meet the demand for real-time simulation.

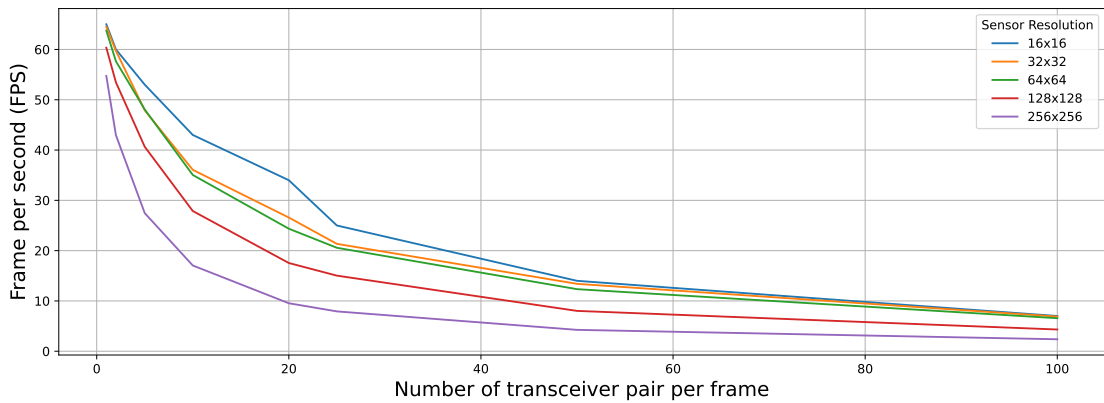


FIGURE 5.23: Variation of frame rate with the number of transceiver pairs performing WiTracing in a single frame under different sensor resolution conditions.

To maximize the real-time **RT** performance and accurately gauge execution efficiency, the **RT** computations are executed within each frame in the aforementioned experiments. In practice, higher frame rates can be achieved with a reasonable task scheduling approach. For example, for real wireless device simulation, the measurement frequency and resolution will be lower, and better simulation performance can be obtained by setting the simulation frequency reasonably. In the previous example, by changing the simulation frequency to 0.1s each time and simulating the signal propagation between 20 pairs of transceivers each time, the frame rate can reach 160 **FPS** under the same hardware conditions. This setting can take into account the real-time simulation of wireless signals as well as dynamic environments at the same time.

5.5 Summary

This study comprehensively demonstrates that WirelessDT stands out in several critical aspects of wireless data simulation and analysis. In the realm of data fidelity, WirelessDT's ability to utilize real-world datasets for accurate simulation and training is exemplary. The system effectively mirrors real-world environments, as evidenced by its superior performance compared to the benchmark path loss model in terms of precision and data distribution closeness.

In terms of practicality, the application of WirelessDT in real-world scenarios, such as wireless localization mobile applications, further cements its utility. The system's adeptness in calibrating and simulating in both small and large-scale virtual environments and its capability to provide real-time, high-quality data to mobile devices showcases its practical applicability in real-world situations. This aspect of WirelessDT is particularly crucial as it bridges the gap between theoretical models and practical, real-world applications, enhancing the reliability and effectiveness of wireless device evaluations.

Lastly, the scalability evaluation of WirelessDT, especially when juxtaposed with established tools like Matlab and Carla, underlines its efficiency and adaptability in data generation. WirelessDT's speed in generating wireless signals, a key metric in scalability, not only

surpasses these traditional tools but also indicates its potential for broader applications in more extensive and complex scenarios.

In conclusion, the comprehensive analysis of WirelessDT across these three domains – fidelity, practicality, and scalability – underscores its remarkable capabilities. This study provides compelling evidence of WirelessDT’s versatility, efficiency, and potential to revolutionize the approach to wireless data simulation and analysis. The implications of these findings are significant, offering promising avenues for future research and practical applications in the wireless technology sector.

CHAPTER 6

Conclusion

6.1 Limitation Discussion

This study integrates the digital twin framework with [RTRT](#) technology, aiming for high-fidelity, real-time wireless signal simulation. It extended the communication module within the UE-based digital twin architecture. This enhancement allows for synchronous real-time data interchange between the physical device's observations and the simulation engine's outputs, facilitating functions like automatic calibration and real-time assessment. However, this study has a limitation in the design of communication security. Specifically, incorporating communication security as a key module needed to be prioritized, potentially leaving the system vulnerable to security breaches. To bolster the system's security in future research, the following four areas are the protectial directions:

- **Data Encryption:** Implement encryption protocols for exchanging data between the twin and physical systems. This measure would mitigate risks associated with data leaks during transmission.
- **Regular Software Updates:** Maintain a routine digital twin system update schedule. Timely installation of security patches is crucial to safeguard against vulnerabilities.
- **Data Anonymization:** Anonymize both twin and sensor data. Concealing device identities is essential for enhancing device security and preventing targeting by malicious entities.
- **Enhanced Access Control:** Establish robust access control mechanisms. This would prevent unauthorized access to the twin system, adding a layer of security.

6.2 Conclusion

The GPU RTRT-based WirelessDT platform has been introduced to address the challenge of accurate wireless signal simulation and real-time evaluation of wireless software applications. This work contributes significantly by providing a comprehensive tool to enhance the design, development, and maintenance of wireless software applications. Through rigorous evaluations, the superiority of WirelessDT in terms of fidelity, practicality, and scalability has been demonstrated. The fidelity evaluation revealed that WirelessDT surpasses the baseline model by achieving remarkable accuracy and similarity in emulating wireless signals. This success is attributed to the implementation of advanced data preprocessing techniques and the incorporation of learning-based calibration processes, ensuring the generation of precise signals. Consequently, WirelessDT offers a reliable platform to simulate high-fidelity wireless environments. Practicality assessment demonstrated the effectiveness of WirelessDT in facilitating real-time performance evaluation of wireless positioning algorithms. By reducing time and effort during the lifecycle of wireless products, WirelessDT shows its value in enabling efficient assessment of wireless software applications. Its ability to support real-time evaluation and provide valuable information contributes to the overall practicality of the platform. Moreover, the scalability evaluation showcased the superiority of WirelessDT compared to existing simulation tools, such as Matlab and Carla, particularly in RT-based wireless sensor simulation. WirelessDT exhibited exceptional performance in managing computationally intensive workloads, enabling large-scale simulations, and supporting real-time analysis. These features position WirelessDT as a preferred choice for researchers and practitioners seeking to simulate and evaluate wireless systems in dynamic environments. In conclusion, the WirelessDT platform significantly advances wireless simulation technology by addressing the challenges of accurate signal simulation and real-time evaluation. Its ability to achieve high fidelity, support real-time performance evaluation, and enable scalability establishes it as a valuable tool for researchers and practitioners in the field. Using WirelessDT, the development and optimization of wireless software applications can be streamlined, leading to improved wireless communication systems and enhanced user experiences.

In upcoming research, our focus will be on integrating WirelessDT into key industrial domains such as smart manufacturing and autonomous driving. Concurrently, this study aim to elevate WirelessDT's fidelity and scalability through advanced algorithmic and model enhancements. Our goal is to broaden the range of sensor types simulated, capturing a vast array of sensors prevalent in industrial settings. This ensures a holistic and precise wireless digital twin platform for researchers and industry professionals.

References

- [Abbas *et al.*2019] Moustafa Abbas, Moustafa Elhamshary, Hamada Rizk, Marwan Torki, and Moustafa Youssef. 2019. WiDeep: Wifi-Based Accurate and Robust Indoor Localization System Using Deep Learning. In *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–10. IEEE.
- [Adege *et al.*2018] Abebe Belay Adege, Hsin-Piao Lin, Getaneh Berie Tarekegn, and Shiann-Shiun Jeng. 2018. Applying Deep Neural Network (DNN) For Robust Indoor Localization in Multi-Building Environment. *Applied Sciences*, 8(7):1062.
- [Alarifi *et al.*2016] Abdulrahman Alarifi, AbdulMalik Al-Salman, Mansour Alsaleh, Ahmad Alnafessah, Suheer Al-Hadhrani, Mai A Al-Ammar, and Hend S Al-Khalifa. 2016. Ultra Wideband Indoor Positioning Technologies: Analysis and Recent Advances. *Sensors*, 16(5):707.
- [ALTAIR2023] ALTAIR. 2023. Radio Coverage Planning with Altair WinProp. <https://web.altair.com/winprop-telecom>.
- [Anderson2003] Harry R Anderson. 2003. *Fixed Broadband Wireless System Design*. John Wiley & Sons.
- [Appel1968] Arthur Appel. 1968. Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, pages 37–45.
- [Apple2013] Apple. 2013. iBeacon.
- [Ayyalasomayajula *et al.*2020] Roshan Ayyalasomayajula, Aditya Arun, Chenfeng Wu, Sanatan Sharma, Abhishek Rajkumar Sethi, Deepak Vasisht, and Dinesh Bharadia. 2020. Deep Learning Based Wireless Localization for Indoor Navigation. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14.
- [Bae and Choi2019] Han Jun Bae and Lynn Choi. 2019. Large-Scale Indoor Positioning using Geomagnetic Field with Deep Neural Networks. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.
- [Banks and Carson1986] Jerry Banks and John S Carson. 1986. Introduction to Discrete-Event Simulation. In *Proceedings of the 18th Conference on Winter Simulation*, pages 17–23.

- [Barricelli *et al.*2019] Barbara Rita Barricelli, Elena Casiraghi, and Daniela Fogli. 2019. A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications. *IEEE Access*, 7:167653–167671.
- [BMWGroup2023] BMWGroup. 2023. This Is How Digital the BMW Ifactory Is. <https://www.bmwgroup.com/en/news/general/2022/bmw-ifactory-digital.html>.
- [Botín-Sanabria *et al.*2022] Diego M Botín-Sanabria, Adriana-Simona Mihaita, Rodrigo E Peimbert-García, Mauricio A Ramírez-Moreno, Ricardo A Ramírez-Mendoza, and Jorge de J Lozoya-Santos. 2022. Digital Twin Technology Challenges and Applications: A Comprehensive Review. *Remote Sensing*, 14(6):1335.
- [Buildmedia2021] Buildmedia. 2021. Wellington Digital Twin. <https://buildmedia.com/work/wellington-digital-twin>.
- [Chapman and Morris2019] IT Chapman and AW Morris. 2019. UKAEA Capabilities to Address the Challenges on the Path to Delivering Fusion Power. *Philosophical Transactions of the Royal Society A*, 377(2141):20170436.
- [Cheng and Zhou2019] Yun Cheng and Taoyun Zhou. 2019. UWB Indoor Positioning Algorithm Based on TDoA Technology. In *2019 10th International Conference on Information Technology in Medicine and Education (ITME)*, pages 777–782. IEEE.
- [De Angelis *et al.*2016] Guido De Angelis, Antonio Moschitta, and Paolo Carbone. 2016. Positioning Techniques in Indoor Environments Based on Stochastic Modeling of Uwb Round-Trip-Time Measurements. *IEEE Transactions on Intelligent Transportation Systems*, 17(8):2272–2281.
- [Dosovitskiy *et al.*2017] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. Carla: An Open Urban Driving Simulator. In *Conference on Robot Learning*, pages 1–16. PMLR.
- [dSpace2023] dSpace. 2023. AURELION: Sensor-Realistic Simulation for ADAS/AD in Real Time. https://www.dspace.com/en/inc/home/products/sw/experimentandvisualization/aurelion_sensor-realistic_sim.cfm.
- [Elsayed and Erol-Kantarci2019] Medhat Elsayed and Melike Erol-Kantarci. 2019. AI-Enabled Future Wireless Networks: Challenges, Opportunities, and Open Issues. *IEEE Vehicular Technology Magazine*, 14(3):70–77.
- [EpicGames2022] EpicGames. 2022. Real-Time Ray Tracing. <https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/RayTracing/>.
- [EpicGames2023] EpicGames. 2023. Unreal Engine. <https://www.unrealengine.com>.

- [Ericsson2021] Ericsson. 2021. Next-Generation Simulation Technology to Accelerate the 5G Journey. <https://www.ericsson.com/en/blog/2021/4/5g-simulation-omniverse-platform>.
- [Ericsson2022] Ericsson. 2022. Digital Twins: What Are They and How Are They Enabling Future Networks? <https://www.ericsson.com/en/blog/2022/3/what-a-re-digital-twins-three-real-world-examples>.
- [Esteves *et al.*2003] João Sena Esteves, Adriano Carvalho, and Carlos Couto. 2003. Generalized Geometric Triangulation Algorithm for Mobile Robot Absolute Self-Localization. In *2003 IEEE International Symposium on Industrial Electronics (Cat. No. 03TH8692)*, volume 1, pages 346–351. IEEE.
- [Fang *et al.*2012] Shih-Hau Fang, Chu-Hsuan Wang, Ting-Yu Huang, Chin-Huang Yang, and Yung-Sheng Chen. 2012. An Enhanced Zigbee Indoor Positioning System With an Ensemble Approach. *IEEE Communications Letters*, 16(4):564–567.
- [Farahsari *et al.*2022] Pooyan Shams Farahsari, Amirhossein Farahzadi, Javad Rezazadeh, and Alireza Bagheri. 2022. A Survey on Indoor Positioning Systems for Iot-Based Applications. *IEEE Internet of Things Journal*, 9(10):7680–7699.
- [Feldmann *et al.*2003] Silke Feldmann, Kyandoghere Kyamakya, Ana Zapater, and Zighuo Lue. 2003. An Indoor Bluetooth-Based Positioning System: Concept, Implementation and Experimental Evaluation. In *International Conference on Wireless Networks*, volume 272.
- [Feng *et al.*2020] Daquan Feng, Chunqi Wang, Chunlong He, Yuan Zhuang, and Xiang-Gen Xia. 2020. Kalman-Filter-based Integration of IMU and UWB for High-Accuracy Indoor Positioning and Navigation. *IEEE Communications Surveys & Tutorials*, 7(4):3133–3146.
- [Gabor *et al.*2016] Thomas Gabor, Lenz Belzner, Marie Kiermeier, Michael Till Beck, and Alexander Neitz. 2016. A Simulation-Based Architecture for Smart Cyber-Physical Systems. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*, pages 374–379. IEEE.
- [Gauxachs Sanz2021] Lluc Gauxachs Sanz. 2021. 3D Procedural Modeling and Game Generation of the UJI Library.
- [Geng *et al.*2021] Chunhua Geng, Traian E Abrudan, Veli-Matti Kolmonen, and Howard Huang. 2021. Experimental Study on Probabilistic Toa and Aoa Joint Localization in Real Indoor Environments. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE.
- [Glaessgen and Stargel2012] Edward Glaessgen and David Stargel. 2012. The Digital Twin Paradigm for Future NASA and US Air Force Vehicles. In *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*, page 1818.

- [Google2015] Google. 2015. Google/Eddystone: Specification for Eddystone, an Open Beacon Format From Google.
- [Grieves and Vickers2017] Michael Grieves and John Vickers. 2017. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*, pages 85–113.
- [Grieves2005a] M Grieves. 2005a. Product Lifecycle Management: Driving the Next Generation of Lean Thinking the McGraw-Hill Co.
- [Grieves2005b] Michael W Grieves. 2005b. Product Lifecycle Management: The New Paradigm for Enterprises. *International Journal of Product Development*, 2(1-2):71–84.
- [Grieves2014] Michael Grieves. 2014. Digital Twin: Manufacturing Excellence Through Virtual Factory Replication. *White Paper*, 1(2014):1–7.
- [Guo *et al.*2019] Xiansheng Guo, Nirwan Ansari, Fangzi Hu, Yuan Shao, Nkrow Raphael Elikplim, and Lin Li. 2019. A Survey on Fusion-Based Indoor Positioning. *IEEE Communications Surveys & Tutorials*, 22(1):566–594.
- [Haines and Akenine-Möller2019] Eric Haines and Tomas Akenine-Möller. 2019. *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other Apis*. Apress.
- [Harrington *et al.*2022] Maria CR Harrington, Chris Jones, and Crissy Peters. 2022. Virtual Nature as a Digital Twin Botanically Correct 3D AR and VR Optimized Low-Polygon and Photogrammetry High-Polygon Plant Models: A Short Overview of Construction Methods. In *ACM SIGGRAPH 2022 Educator’s Forum*, pages 1–2.
- [Hazas and Hopper2006] Mike Hazas and Andy Hopper. 2006. Broadband Ultrasonic Location Systems for Improved Indoor Positioning. *IEEE Transactions on Mobile Computing*, 5(5):536–547.
- [He and Chan2015] Suining He and S-H Gary Chan. 2015. Wi-Fi Fingerprint-Based Indoor Positioning: Recent Advances and Comparisons. *IEEE Communications Surveys & Tutorials*, 18(1):466–490.
- [He *et al.*2019] Danping He, Bo Ai, Ke Guan, Longhe Wang, Zhangdui Zhong, and Thomas Kürner. 2019. The Design and Applications of High-Performance Ray-Tracing Simulation Platform for 5G and Beyond Wireless Communications: A Tutorial. *IEEE Communications Surveys & Tutorials*, 21(1):10–27.
- [Hempstead and Worthington2005] Colin Hempstead and William Worthington. 2005. *Encyclopedia of 20th-Century Technology*. Routledge.
- [Höflinger *et al.*2012] Fabian Höflinger, Rui Zhang, Joachim Hoppe, Amir Bannoura, Leonhard M Reindl, Johannes Wendeberg, Manuel Bühner, and Christian Schindelbauer. 2012. Acoustic Self-Calibrating System for Indoor Smartphone Tracking (Assist). In *2012 international conference on indoor positioning and indoor navigation (IPIN)*, pages 1–9.

IEEE.

- [Hofmann1990] Georg Rainer Hofmann. 1990. Who Invented Ray Tracing? A Historical Remark. *The Visual Computer*, 6(3):120–124.
- [HOK2022] HOK. 2022. Centre Block Rehabilitation. <https://www.hok.com/projects/view/centre-block-rehabilitation/>.
- [Huang and Chan2011] Chih-Ning Huang and Chia-Tai Chan. 2011. Zigbee-based Indoor Location System by K-Nearest Neighbor Algorithm With Weighted RSSI. *Procedia Computer Science*, 5:58–65.
- [Huang et al.2023] He Huang, Jianfei Yang, Xu Fang, Hao Jiang, and Lihua Xie. 2023. Fusion of WiFi and IMU Using Swarm Optimization for Indoor Localization. In *Machine Learning for Indoor Localization and Navigation*, pages 133–157. Springer.
- [Jang and Kim2018] Beakcheol Jang and Hyunjung Kim. 2018. Indoor Positioning Technologies Without Offline Fingerprinting Map: A Survey. *IEEE Communications Surveys & Tutorials*, 21(1):508–525.
- [Javaid and Haleem2023] Mohd Javaid and Abid Haleem. 2023. Digital Twin Applications Toward Industry 4.0: A Review. *Cognitive Robotics*.
- [Kaune2012] Regina Kaune. 2012. Accuracy Studies for TDoA and ToA Localization. In *2012 15th International Conference on Information Fusion*, pages 408–415. IEEE.
- [Keinert et al.2015] Benjamin Keinert, Matthias Innmann, Michael Sanger, and Marc Stamminger. 2015. Spherical Fibonacci Mapping. *ACM Transactions on Graphics (TOG)*, 34(6):1–7.
- [Kok et al.2015] Manon Kok, Jeroen D Hol, and Thomas B Schon. 2015. Indoor Positioning Using Ultrawideband and Inertial Measurements. *IEEE Transactions on Vehicular Technology*, 64(4):1293–1303.
- [Lai et al.2022] Zhongzheng Lai, Dong Yuan, Wei Bao, Yu Zhang, and Bing Bing Zhou. 2022. DeepWiSim: A Wireless Signal Simulator for Automatic Deep Learning. In *Proceedings of the SIGCOMM ’22 Poster and Demo Sessions*, pages 13–15.
- [Lai et al.2023] Zhongzheng Lai, Dong Yuan, Huaming Chen, Yu Zhang, and Wei Bao. 2023. WirelessDT: A Digital Twin Platform for Real-Time Evaluation of Wireless Software Applications. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 146–150. IEEE.
- [Lam and Little2019] Emily W Lam and Thomas DC Little. 2019. Indoor 3D Localization With Low-Cost Lifi Components. In *2019 Global LIFI Congress (GLC)*, pages 1–6. IEEE.
- [Lee et al.2018] Namkyoung Lee, Sumin Ahn, and Dongsoo Han. 2018. AMID: Accurate Magnetic Indoor Localization Using Deep Learning. *Sensors*, 18(5):1598.

- [Ling *et al.*1989] Hao Ling, R-C Chou, and S-W Lee. 1989. Shooting and Bouncing Rays: Calculating the Rcs of an Arbitrarily Shaped Cavity. *IEEE Transactions on Antennas and Propagation*, 37(2):194–205.
- [Liu and Wang2019] KJ Ray Liu and Beibei Wang. 2019. *Wireless AI: Wireless Sensing, Positioning, Iot, and Communications*. Cambridge University Press.
- [Liu *et al.*2007] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. 2007. Survey of Wireless Indoor Positioning Techniques and Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1067–1080.
- [Liu *et al.*2020] Fen Liu, Jing Liu, Yuqing Yin, Wenhan Wang, Donghai Hu, Pengpeng Chen, and Qiang Niu. 2020. Survey on Wifi-Based Indoor Positioning Techniques. *IET communications*, 14(9):1372–1383.
- [Ma *et al.*2019] Xin Ma, Fei Tao, Meng Zhang, Tian Wang, and Ying Zuo. 2019. Digital Twin Enhanced Human-Machine Interaction in Product Lifecycle. *Procedia CIRP*, 83:789–793.
- [MathWorks2023] MathWorks. 2023. MATLAB. <https://ww2.mathworks.cn>.
- [Maxwell1873] James Clerk Maxwell. 1873. *A Treatise on Electricity and Magnetism*, volume 1. Clarendon Press.
- [Mercedes-BenzGroup2023] Mercedes-BenzGroup. 2023. Industry 4.0 and the Networked Factory. <https://group.mercedes-benz.com/innovation/case/connectivity/industry-4-0.html>.
- [Meyer *et al.*2010] Quirin Meyer, Jochen Süßmuth, Gerd Sußner, Marc Stamminger, and Günther Greiner. 2010. On Floating-Point Normal Vectors. In *Computer Graphics Forum*, volume 29, pages 1405–1409. Wiley Online Library.
- [Mihai *et al.*2022] Stefan Mihai, Mahnoor Yaqoob, Dang V Hung, William Davis, Praveer Towakel, Mohsin Raza, Mehmet Karamanoglu, Balbir Barn, Dattaprasad Shetve, Raja V Prasad, et al. 2022. Digital Twins: A Survey on Enabling Technologies, Challenges, Trends and Future Prospects. *IEEE Communications Surveys & Tutorials*.
- [Morohin and Totsky2021] DV Morohin and AA Totsky. 2021. Indoor Ultrasonic Positioning System for Mobile Objects. In *IOP Conference Series: Materials Science and Engineering*, volume 1047, page 012118. IOP Publishing.
- [Neskovic *et al.*2000] Aleksandar Neskovic, Natasa Neskovic, and George Paunovic. 2000. Modern Approaches in Modeling of Mobile Radio Systems Propagation Environment. *IEEE Communications Surveys & Tutorials*, 3(3):2–12.
- [Ni *et al.*2022] Jiazhi Ni, Fusang Zhang, Jie Xiong, Qiang Huang, Zhaoxin Chang, Junqi Ma, BinBin Xie, Pengsen Wang, Guangyu Bian, Xin Li, et al. 2022. Experience: Pushing Indoor Localization From Laboratory to the Wild. In *Proceedings of the 28th Annual International Conference on Mobile Computing and Networking*, pages 147–157.

- [Nicodemus1965] Fred E Nicodemus. 1965. Directional Reflectance and Emissivity of an Opaque Surface. *Applied Optics*, 4(7):767–775.
- [NVIDIA2022] NVIDIA. 2022. Develop on NVIDIA Omniverse. <https://developer.nvidia.com/nvidia-omniverse-platform>.
- [NVIDIA2023a] NVIDIA. 2023a. Digital Twins - Harness the Power of Large-Scale Simulation for Industrial and Scientific Applications. <https://www.nvidia.com/en-us/omniverse/solutions/digital-twins/>.
- [NVIDIA2023b] NVIDIA. 2023b. Ray Tracing | NVIDIA Developer. <https://developer.nvidia.com/discover/ray-tracing>.
- [Piascik *et al.*2010] R Piascik, John Vickers, Dave Lowry, Steve Scotti, Jeff Stewart, and Anthony Calomino. 2010. Technology Area 12: Materials, Structures, Mechanical Systems, and Manufacturing Road Map. *NASA Office of Chief Technologist*, pages 15–88.
- [Poynting1884] John Henry Poynting. 1884. XV. On the Transfer of Energy in the Electromagnetic Field. *Philosophical Transactions of the Royal Society of London*, pages 343–361.
- [Qi *et al.*2021] Qinglin Qi, Fei Tao, Tianliang Hu, Nabil Anwer, Ang Liu, Yongli Wei, Lihui Wang, and AYC Nee. 2021. Enabling Technologies and Tools for Digital Twin. *Journal of Manufacturing Systems*, 58:3–21.
- [RaspberryPi2023] RaspberryPi. 2023. Raspberry Pi. <https://www.raspberrypi.com/>.
- [Remcom2023] Remcom. 2023. Wireless EM Propagation Software - Wireless InSite — Remcom. <https://www.remcom.com/wireless-insite-em-propagation-software>.
- [Riley and Henderson2010] George F Riley and Thomas R Henderson. 2010. The NS-3 Network Simulator. In *Modeling and Tools for Network Simulation*, pages 15–34. Springer.
- [Rindel2000] Jens Holger Rindel. 2000. The Use of Computer Modeling in Room Acoustics. *Journal of Vibroengineering*, 3(4):219–224.
- [Saab and Nakad2010] Samer S Saab and Zahi S Nakad. 2010. A Standalone RFID Indoor Positioning System Using Passive Tags. *IEEE Transactions on Industrial Electronics*, 58(5):1961–1970.
- [Schneider2015] Terry Schneider. 2015. From Atoms to Airplanes. <https://compassmag.3ds.com/from-atoms-to-airplanes/>.
- [Seco *et al.*2010] Fernando Seco, Christian Plagemann, Antonio R Jiménez, and Wolfram Burgard. 2010. Improving Rfid-Based Indoor Positioning Accuracy Using Gaussian Processes. In *2010 International Conference on Indoor Positioning and Indoor Navigation*,

- pages 1–8. IEEE.
- [Sertatıl *et al.*2012] Cem Sertatıl, Mustafa A Altinkaya, and Kosai Raoof. 2012. A Novel Acoustic Indoor Localization System Employing CDMA. *Digital Signal Processing*, 22(3):506–517.
- [Shafto *et al.*2012] Mike Shafto, Mike Conroy, Rich Doyle, Ed Glaessgen, Chris Kemp, Jacqueline LeMoigne, and Lui Wang. 2012. Modeling, Simulation, Information Technology & Processing Roadmap. *National Aeronautics and Space Administration*, 32(2012):1–38.
- [Shah *et al.*2018] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2018. Airsim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics: Results of the 11th International Conference*, pages 621–635. Springer.
- [Shang *et al.*2015] Jianga Shang, Xuke Hu, Fuqiang Gu, Di Wang, Shengsheng Yu, et al. 2015. Improvement Schemes for Indoor Mobile Location Estimation: A Survey. *Mathematical Problems in Engineering*, 2015.
- [Shchekotov2014] Maxim Shchekotov. 2014. Indoor Localization Method Based on Wi-Fi Trilateration Technique. In *Proceeding of the 16th Conference of Fruct Association*, pages 177–179.
- [Siemens2017] M Siemens. 2017. For a Digital Twin of the Grid: Siemens Solution Enables a Single Digital Grid Model of the Finnish Power System. <https://assets.new.siemens.com/siemens/assets/api/uuid:09c20834-4ed4-49d8-923d-ebcc541cab37/inno2017-digitaltwin-e.pdf>.
- [Simulia2020] Dassault Systèmes Simulia. 2020. CST Studio Suite. <https://www.cati.com/wp-content/uploads/2022/02/cst-products-list-cati-2022.pdf>.
- [Singh *et al.*2021] Maulshree Singh, Evert Fuenmayor, Eoin P Hinchy, Yuansong Qiao, Niall Murray, and Declan Devine. 2021. Digital Twin: Origin to Future. *Applied System Innovation*, 4(2):36.
- [SYNOPSIS2023] SYNOPSIS. 2023. What is Ray Tracing. <https://www.synopsys.com/glossary/what-is-ray-tracing.html>.
- [Tao *et al.*2022] Fei Tao, Bin Xiao, Qinglin Qi, Jiangfeng Cheng, and Ping Ji. 2022. Digital Twin Modeling. *Journal of Manufacturing Systems*, 64:372–389.
- [Taylor2023] Petroc Taylor. 2023. Forecast number of Mobile Users Worldwide 2020-2025, Jan.
- [Toasa *et al.*2021] Fabricio A Toasa, Luis Tello-Oquendo, Carlos R Peñafiel-Ojeda, and Giovanni Cuzco. 2021. Experimental Demonstration for Indoor Localization Based on Aoa of Bluetooth 5.1 Using Software Defined Radio. In *2021 IEEE 18th Annual Consumer*

- Communications & Networking Conference (CCNC)*, pages 1–4. IEEE.
- [Torres-Sospedra *et al.*2020] Joaquín Torres-Sospedra, Darwin Quezada Gaibor, Antonio R. Jiménez, Antoni Pérez-Navarro, and Fernando Seco. 2020. Datasets and Supporting Materials for the IPIN 2020 Competition Track 3 (Smartphone-based, off- site), December.
- [Tranter2004] William H Tranter. 2004. *Principles of Communication Systems Simulation With Wireless Applications*. United States of America.
- [Tsingos *et al.*2001] Nicolas Tsingos, Thomas Funkhouser, Addy Ngan, and Ingrid Carlbom. 2001. Modeling Acoustics in Virtual Environments Using the Uniform Theory of Diffraction. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 545–552.
- [Unity2023] Unity. 2023. Top 10 Use Cases for Digital Twins.
- [UnityTechnologies2023] UnityTechnologies. 2023. Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine. <https://unity.com>.
- [Varea García2020] Rebeca Varea García. 2020. Location Indoors.
- [Varga2010] Andras Varga. 2010. OMNeT++. In *Modeling and Tools for Network Simulation*, pages 35–59. Springer.
- [Veach and Guibas1995] Eric Veach and Leonidas J Guibas. 1995. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 419–428.
- [Vouse2021] Vouse. 2021. Digital Twin for Changi Airport. <https://www.vouse.co/gallery-changi-aerodrome-mobile>.
- [Wang *et al.*2010] Shangbo Wang, Andreas Waadt, Admir Burnic, Dong Xu, Christian Kocks, Guido H Bruck, and Peter Jung. 2010. System Implementation Study on Rssi Based Positioning in UWB Networks. In *2010 7th International Symposium on Wireless Communication Systems*, pages 36–40. IEEE.
- [Wang *et al.*2013] Yapeng Wang, Xu Yang, Yutian Zhao, Yue Liu, and Laurie Cuthbert. 2013. Bluetooth Positioning Using Rssi and Triangulation Methods. In *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*, pages 837–842. IEEE.
- [Wang *et al.*2016] Bang Wang, Qiuyun Chen, Laurence T Yang, and Han-Chieh Chao. 2016. Indoor Smartphone Localization via Fingerprint Crowdsourcing: Challenges and Approaches. *IEEE Wireless Communications*, 23(3):82–89.
- [Wang *et al.*2023] Yuntao Wang, Zhou Su, Shaolong Guo, Minghui Dai, Tom H Luan, and Yiliang Liu. 2023. A Survey on Digital Twins: Architecture, Enabling Technologies, Security and Privacy, and Future Prospects. *IEEE Internet of Things Journal*.

- [Wolski2011] Andrzej Wolski. 2011. Theory of Electromagnetic Fields. *Arxiv Preprint arxiv:1111.4354*.
- [Wu *et al.*2021] Yiwen Wu, Ke Zhang, and Yan Zhang. 2021. Digital Twin Networks: A Survey. *IEEE Internet of Things Journal*, 8(18):13789–13804.
- [Xiao *et al.*2016] Jiang Xiao, Zimu Zhou, Youwen Yi, and Lionel M Ni. 2016. A Survey on Wireless Indoor Localization From the Device Perspective. *ACM Computing Surveys (CSUR)*, 49(2):1–31.
- [Xiao *et al.*2017] Chao Xiao, Daiqin Yang, Zhenzhong Chen, and Guang Tan. 2017. 3-D Ble Indoor Localization Based on Denoising Autoencoder. *IEEE Access*, 5:12751–12760.
- [Yang and Shao2015] Chouchang Yang and Huai-Rong Shao. 2015. Wifi-nased Indoor Positioning. *IEEE Communications Magazine*, 53(3):150–157.
- [Yang *et al.*2013] Zheng Yang, Zimu Zhou, and Yunhao Liu. 2013. From RSSI to CSI: Indoor Localization via Channel Response. *ACM Computing Surveys (CSUR)*, 46(2):1–32.
- [Yun and Iskander2015] Zhengqing Yun and Magdy F Iskander. 2015. Ray Tracing for Radio Propagation Modeling: Principles and Applications. *IEEE Access*, 3:1089–1100.
- [Zafari *et al.*2019] Faheem Zafari, Athanasios Gkelias, and Kin K Leung. 2019. A Survey of Indoor Localization Systems and Technologies. *IEEE Communications Surveys & Tutorials*, 21(3):2568–2599.
- [Zaidi *et al.*2018] Ali Zaidi, Fredrik Athley, Jonas Medbo, Ulf Gustavsson, Giuseppe Durisi, and Xiaoming Chen. 2018. *5G Physical Layer: Principles, Models and Technology Components*. Academic Press.
- [Zayets and Steinbach2017] Alexandra Zayets and Eckehard Steinbach. 2017. Robust Wifi-Based Indoor Localization Using Multipath Component Analysis. In *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8. IEEE.
- [Zhang *et al.*2019] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. 2019. Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Communications Surveys & Tutorials*, 21(3):2224–2287.
- [Zheng *et al.*2018] Yang Zheng, Min Sheng, Junyu Liu, and Jiandong Li. 2018. Exploiting AoA Estimation Accuracy for Indoor Localization: A Weighted AoA-Based Approach. *IEEE Wireless Communications Letters*, 8(1):65–68.
- [Zhou *et al.*2017] Cheng Zhou, Jiazheng Yuan, Hongzhe Liu, and Jing Qiu. 2017. Bluetooth Indoor Positioning Based on RSSI and Kalman Filter. *Wireless Personal Communications*, 96:4115–4130.