

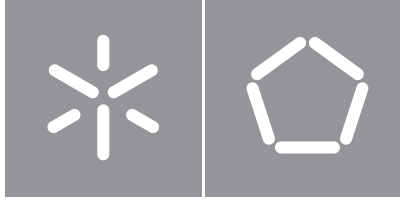


Universidade do Minho

Escola de Engenharia

Gonçalo Araújo Martins

Home Partitions as Interactive Surfaces



Universidade do Minho

Escola de Engenharia

Gonçalo Araújo Martins

Home Partitions as Interactive Surfaces

Master's Dissertation

Master's in Industrial Electronics and Computers Engineering

Work supervised by

Professor Doutor Paulo Cardoso

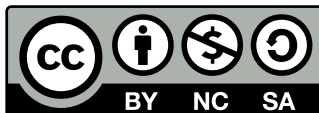
COPYRIGHT AND TERMS OF USE OF THIS WORK BY A THIRD PARTY

This is academic work that can be used by third parties as long as internationally accepted rules and good practices regarding copyright and related rights are respected.

Accordingly, this work may be used under the license provided below.

If the user needs permission to make use of the work under conditions not provided for in the indicated licensing, they should contact the author through the RepositóriUM of Universidade do Minho.

License granted to the users of this work



**Creative Commons Atribuição-NãoComercial-Compartilhalgual 4.0 Internacional
CC BY-NC-SA 4.0**

<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.pt>

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

Acknowledgements

This dissertation marks the end of an important phase of my life. These years brought steep obstacles and difficult challenges, but also offered me the opportunity to overcome them and grow. In reaching this accomplishment, I cannot forget the support and help received from everyone who lent me their time, patience, and kindness.

A special thank you to my supervisor, Professor Paulo Cardoso, for the guidance and trust I felt throughout our work together.

To my family and my girlfriend, thank you for your unwavering support during this time, and for giving me the privilege of always having a home and comfort to return to.

To my close friends, a heartfelt thank you for the immense help you offered me throughout these years. All the experiences we shared are precious memories that I am fortunate to carry with me, even the painful ones.

To all of you, thank you!

Abstract

This dissertation investigates the potential of domestic partition walls as interactive surfaces within the context of smart environments and human–building interaction. The work focuses on the design and development of a capacitive gesture recognition module intended for unobtrusive integration into residential spaces, enabling contactless control of different home automation systems.

The proposed solution combines capacitive proximity sensing with digital signal processing and an embedded machine learning model to achieve robust real-time gesture recognition. The system architecture emphasizes modularity, privacy, and computational efficiency, avoiding the use of cameras or wearable devices while remaining compatible with resource-constrained embedded platforms.

The implementation covers the development of the sensing hardware, embedded firmware for signal acquisition and preprocessing, deployment of a lightweight classifier on a microcontroller, and communication with an external actuation system. As a proof of concept, the gesture recognition module was integrated with an LED-based partition wall, demonstrating the decoupling between the interaction mechanism and the controlled actuators.

Experimental validation includes sensor front-end characterization, timing analysis of the processing pipeline, classifier performance evaluation, and field testing under realistic interaction conditions. The results show that the system achieves reliable gesture recognition within practical interaction distances, satisfies real-time constraints, and exhibits robust behavior in the presence of noise and environmental variability. Overall, this work demonstrates the feasibility of embedding capacitive gesture-based interfaces directly into architectural elements, contributing to the development of more natural, modular, and user-centric smart home environments.

Keywords: Human–Building Interaction; Smart Environments; Gesture Recognition; Capacitive Sensing; Embedded Machine Learning; Natural Interfaces

Resumo

Esta dissertação explora o potencial de paredes divisórias domésticas como superfícies interativas, inseridas no contexto de ambientes inteligentes e da interação humano-edifício. O trabalho centra-se no desenvolvimento de um módulo de reconhecimento de gestos baseado em detecção capacitiva, concebido para ser integrado de forma discreta em ambientes residenciais e utilizado como interface de controlo sem contacto para diferentes sistemas domésticos.

O sistema proposto combina um conjunto de eléktodos capacitivos com processamento digital de sinal e um modelo de aprendizagem automática embarcado, permitindo o reconhecimento robusto de gestos manuais em tempo real. A arquitetura foi desenhada com foco na modularidade, privacidade e eficiência computacional, evitando o recurso a câmaras ou dispositivos vestíveis e garantindo compatibilidade com plataformas embebidas de recursos limitados.

A implementação inclui o desenvolvimento do hardware do módulo, o firmware para aquisição e pré-processamento de sinais, a integração de um classificador leve executado num microcontrolador, e a comunicação com um sistema externo de atuação. Como prova de conceito, o módulo foi integrado num sistema de parede divisória com iluminação LED, demonstrando a separação entre o mecanismo de interação e os atuadores controlados.

A validação experimental abrange a caracterização do front-end sensorial, a análise temporal do pipeline de processamento, a avaliação do desempenho do classificador e testes em condições reais de utilização. Os resultados obtidos demonstram que a solução é capaz de reconhecer gestos de forma fiável em distâncias de interação práticas, cumprindo requisitos de tempo real e apresentando um comportamento robusto face a ruído e variações ambientais. O trabalho confirma, assim, a viabilidade de integrar interfaces gestuais capacitivas diretamente em elementos arquitetónicos, contribuindo para o desenvolvimento de ambientes domésticos mais naturais, modulares e centrados no utilizador.

Palavras-Chave: Interação Humano-Edifício; Ambientes Inteligentes; Reconhecimento de Gestos; Detecção Capacitiva; Aprendizagem Automática Embarcada; Interfaces Naturais

Contents

List of Figures	x
List of Tables	xii
Acronyms	xiii
1 Introduction	1
1.1 Contextualization	1
1.1.1 History of Home Automation	1
1.1.2 Human-Building Interactions	2
1.1.3 Interactive Partitions in the Context of Home Automation	3
1.2 Project Evolution and Current Scope	4
1.3 Motivation	6
1.4 Objectives	6
1.5 Document Structure	7
2 Literature Review	8
2.1 Smart Environments and Ambient Intelligence	8
2.1.1 From Ubiquitous Computing to Ambient Intelligence	8
2.1.2 Characteristics of Smart Environments	9
2.1.3 Human-Building Interaction and Natural Interfaces	9
2.2 Gesture-Based Human-Computer Interaction	10
2.2.1 Gesture Recognition in HCI	10
2.2.2 Technologies for Gesture Recognition	11
2.3 Capacitive Sensing for Gesture Interfaces	12
2.3.1 Principles of Capacitive Sensing	12
2.3.2 Capacitive Sensor Design	15
2.3.3 Real-World Applications	19
2.4 Machine Learning for Gesture Recognition	19

2.4.1	Motivation for Machine Learning (ML) in Capacitive Sensing	19
2.4.2	Learning-by-Example Paradigms	20
2.4.3	Embedded and Edge Deployment	21
2.5	Synthesis and Research Gap	23
2.5.1	Synthesis of Reviewed Literature	23
2.5.2	Identified Gaps in Current Research and Practice	24
2.5.3	Concluding Remarks	25
3	Design and Integration of the Gesture Recognition System	26
3.1	System Overview	26
3.1.1	System Purpose and Scope	26
3.2	Gesture Recognition Module Design	28
3.2.1	Requirements Analysis	28
3.2.2	System Overview	30
3.2.3	Hardware Architecture	31
3.2.4	Software Architecture	40
3.2.5	Embedded Machine Learning Integration	48
3.3	Integration with Actuator Systems	54
3.3.1	Target Actuation System: LED Wall Architecture	54
3.3.2	Interface Between Gesture Module and Actuator System	56
3.3.3	Example Gesture-to-Action Scenarios	59
3.3.4	Generalization to Other Actuators (Beyond LED Walls)	60
3.4	Design Summary	60
3.4.1	Architecture at a Glance	61
3.5	Design Validation and Summary	61
3.5.1	What the Design Establishes	61
3.5.2	Why These Choices	61
3.5.3	Interfaces Summarized	62
3.5.4	Constraints and Assumptions	62
3.5.5	Deferred to Implementation	62
4	Implementation of the Gesture Recognition Module and Prototype System	63
4.1	Gesture Module Hardware and Physical Structure	63
4.1.1	Layered Structure (Back Panel + Front Panel)	64
4.1.2	Protoboard and Circuit (Back Panel)	64
4.1.3	Sensing Surface with 3D-Printed Electrodes and Shield Rings (Front Panel)	65
4.1.4	3D-Printed Outer Structure and Front Panel Mount	67

4.2	Firmware and ML Implementation on the ESP32	70
4.2.1	Codebase Overview and Build Tooling	70
4.2.2	Sensor Sampling and Processing	72
4.2.3	Preprocessing for Machine Learning	81
4.2.4	Model Training, Quantization, and Embedded Deployment	84
4.2.5	Communication protocol to the Raspberry Pi	89
5	Verification and Results	92
5.1	Sensor and Front-End Characterization	92
5.1.1	Idle Noise, Drift, and Minimum Detectable Change	92
5.2	Acquisition Timing	94
5.2.1	Sampling Cadence at 100 Hz (Raw)	94
5.2.2	Post-Pipeline Emission Cadence at 33 Hz	94
5.3	Impact of the Filtering Pipeline	95
5.4	Classifier Performance — Summary	97
5.5	Classifier Performance (Offline)	97
5.5.1	Embedded Runtime (ESP32)	97
5.6	Field Test by Distance	98
6	Conclusion	101
6.1	Summary of Contributions	101
6.2	Evaluation of Objectives and Requirements	102
6.3	Limitations	102
6.4	Future Work	103
6.5	Final Remarks	103
	Bibliography	104

List of Figures

1	Self vs Mutual Capacitance	14
2	Capacitance vs Distance for Various Sensor Area Sizes	17
3	Linear Slide Electrode	17
4	Interdigitated Electrodes	18
5	System Overview	27
6	High-level conceptual architecture of the gesture recognition module	30
7	Functional Diagram of the FDC1004, based on the datasheet [33]	34
8	Electrodes and Sensing Surface Design	38
9	ESP32 Illustration	39
10	Electrical Connection Diagram	40
11	Software Architecture Layers Diagram	41
12	Main Loop Flowchart	42
13	Setup process flowchart	43
14	Data Acquisition Algorithm	45
15	Sliding Ring Buffer Diagram	46
16	State Diagram	46
17	TCP Send Gesture Event	48
18	Target System Architecture	55
19	Integration of the Gesture Recognition Module (GRM) into the existing control stack. The new Gesture Module Process mirrors other panel processes and uses the same gateway queue.	56
20	Algorithmic representation of the Gesture Module Process on the Raspberry Pi (RPi). . . .	58
21	Layered structure of the module	64
22	Left: top side with ESP32 and FDC1004 breakout, Right: Connections between ESP32 and FDC1004	65
23	Front panel. Left: front side with round electrodes and shield rings. Right: underside of the sensing surface	66

24	3D CAD of electrode and ring-shield forms used to shape the copper tape and ensure repeat- able geometry.	67
25	3D CAD of the outer structure	68
26	Real assembly. Left: module mounted with sliding cover in place. Right: Female screw for tripod	68
27	Tripod mount piece	69
28	Top-level repository layout.	70
29	Firmware tree: headers in <code>include/</code> , sources in <code>src/</code> , FDC1004 driver in <code>lib/</code>	71
30	Testing utilities for capture, dataset build, training, conversion, and visualization.	71
31	Timing diagram of the firmware sampling and processing pipeline.	73
32	Histogram of inter-sample intervals	74
33	Baseline-subtracted raw capacitive signal over 60 s.	76
34	Welch Power Spectral Density (PSD) of the raw capacitive signal.	76
35	Comparison between Gesture Signatures, at the same distance(2cm). Left: Raw Signal; Right: Filtered Signal	79
36	Model Architecture	84
37	Learning curves for the selected seed: fast convergence, stable validation beyond early epochs (seed 456).	85
38	Normalized confusion matrix on the test set (seed 456). Off-diagonals are confined to <i>{down, left}</i> and <i>{right, none}</i> at ~3%.	86
39	ESP32 boot and calibration.	88
40	ESP32 runtime excerpt with multiple gestures.	88
41	GestureP server startup on the Raspberry Pi.	89
42	Application framing. The server first reads two bytes to obtain the JavaScript Object Notation (JSON) payload length, then reads exactly that many bytes.	90
43	Server receive loop in operation.	91
44	Raw time-domain output of the four capacitive channels during a 120 s idle acquisition at 100 Hz, prior to any baseline removal or digital conditioning.	93
45	PSD of a representative capacitive channel under idle conditions and during a representative gesture, computed using Welch's method (Hann window, 256-sample segments, 50% overlap).	94
46	Inter-emit interval histogram after the full pipeline (≈ 33 Hz).	95
47	Evolution of a representative swipe gesture (channel 1) through successive processing stages, from baseline-stabilized input to final filtered output.	96

List of Tables

1	Dielectric Constants of Common Materials	13
2	Proximity sensing range based on sensor area size, using data from [34].	36
3	Design invariants and tunable parameters. Concrete values are finalized in the Implementation chapter.	49
4	Key tensor shapes (symbolic). Final K, f_1, f_2, h are fixed in Implementation.	53
5	Gesture-to-command mapping for the LED wall	59
6	Back panel connections (as built).	65
7	Acquisition parameters (fixed in firmware).	75
8	DSP configuration used for gesture data collection.	81
9	Preprocessing artefacts.	81
10	Validation F1 across seeds. Dispersion is small; seed 456 is best.	86
11	Per-class test metrics for the selected seed.	86
12	Accuracy under temporal rolls (test set, seed 456).	87
13	Gesture event payload (ESP32 $\rightarrow$ Raspberry Pi).	90
14	Per-channel idle noise, drift, and noise-limited resolution from raw data at 100 Hz (duration 120 s).	93
15	Inter-emit interval statistics at ≈ 33 Hz. Nominal target ≈ 30.0 ms.	95
16	Offline headline metrics (held-out test set; seed selected in Table 10).	97
17	Embedded deployment footprint on ESP32 (TFLM INT8).	98
18	On-device runtime latency on ESP32 (TFLM INT8).	98
19	Gesture detection rate under varying hand–sensor distances.	99
20	Average confidence of valid detections vs distance.	99

Acronyms

AC Alternating Current

AmI Ambient Intelligence

ANNs Artificial Neural Networks

API Application Programming Interface

AR Augmented Reality

CDC Capacitance-to-Digital Converter

CNN Convolutional Neural Network

CNNs Convolutional Neural Networks

EF Electric Field

EMI Electromagnetic Interference

FSM Finite State Machine

GNNs Graph Neural Networks

GRM Gesture Recognition Module

GUI Graphical User Interface

HBI Human-Building Interaction

HCI Human-Computer Interaction

HMI Human Machine Interface

HMMs Hidden Markov Models

HTTP Hypertext Transfer Protocol

HVAC Heating, Ventilation and Air Conditioning

IC Integrated Circuit

IoT Internet of Things

JSON JavaScript Object Notation

LED Light-Emitting Diode

LSTM Long Short-Term Memory

MCU Microcontroller

ML Machine Learning

MQTT Message Queuing Telemetry Transport

PCB Printed Circuit Board

PSD Power Spectral Density

RAM Random Access Memory

RNNs Recurrent Neural Networks

RPi Raspberry Pi

SDK Software Development Kit

SoC System on Chip

SVMs Support Vector Machines

TCP Transmission Control Protocol

TFLM TensorFlow Lite for Microcontrollers

UART Universal Asynchronous Receiver/Transmitter

VR Virtual Reality

Chapter 1

Introduction

1.1 Contextualization

Modern home spaces have witnessed a dramatic transformation in recent times, as new technology permeates nearly every aspect, turning what were once inert structures into dynamic, intelligent living spaces. This transformation is primarily driven by the concept of home automation, which initially emerged by integrating automotive technology into household appliances and permanently changed how people interact with their homes. Over the years, this concept has expanded beyond initial control over lighting and [Heating, Ventilation and Air Conditioning \(HVAC\)](#), now encompassing a wide range of devices and appliances to create a "smart" ecosystem that adapts to user preferences and the environment.

1.1.1 History of Home Automation

Home automation as a concept can be traced back to the early 20th century, even though it was named differently. The appearance of electrical systems caused the first basic automation to be introduced in the form of automatic thermostats or timers to control lights and heating. With these advances, several basic home appliances were introduced (washing machines, toasters, vacuum cleaners), which, though operated manually, still marked the beginning of home environment automation.

It wasn't until the 1970s that the field of home automation saw major improvements. However, by that time microelectronics had developed significantly and brought with it the possibility of programmable systems. With this, some communication protocols appeared as the first steps in general purpose home automation network technology, enabling communication between different devices through the existing electrical wiring. This brought a new level of sophistication, with revolutionary functionalities like remote control of lighting and appliances.

Between the mid 90's and early 2000's digital technology was being introduced in the home automation scenario, with this came the shift from wired to wireless technologies that enabled higher flexibility and scalability to home systems.

The 2000's brought the era of [Internet of Things \(IoT\)](#) with the rise of Wi-Fi-enabled devices and smart-phones that transformed every possible gadget into a device to be controlled and part of a “smart home” system. Since then, the IoT idea has expanded to be a common aspect of modern-day home environments, with integration of smart hubs and voice assistance, home automation is reaching the point of becoming a seamless part of daily life scenario. With this came the shift from wired to wireless technologies that enabled higher flexibility and scalability to home systems.

1.1.2 Human-Building Interactions

[Human-Building Interaction \(HBI\)](#) refers to the relationship between individuals and the built environment that surrounds them. This concept has a focus on the integration of architecture design and technology to innovate this interaction, encompassing themes like [Human-Computer Interaction \(HCI\)](#), and [Ambient Intelligence \(AmI\)](#) to create spaces that adapt to human needs, enhance individuals' experience, and promote well-being.

[HBI](#) can be explored through various distinct platforms, however, the two most impactful are physical interactions and spatial interactions:

- Physical Interaction involves tangible interfaces that can respond to human input, enabling control and interactivity to the structure of a material element. Examples include gesture recognition or touch-responsive surfaces.
- The spatial aspect focuses on the adaptability of an interior space. Through dynamic space design, an environment can adjust to human needs, behavior or movement, improving functionality and comfort.

In the context of today's “smart homes”, [HBI](#) brings a user-centric design approach that prioritizes individual preferences, behaviors, and needs. This approach has an impact on the home environment by shifting interactivity from a one-size-fits-all model to one tailored to specific user needs and preferences, enhancing the comfort and efficiency of home spaces

Traditionally, walls in a house offered limited innovation potential beyond climate and acoustic functions, with no interactivity. However, in “smart homes,” partition walls present significant opportunities for innovation by integrating interactive elements within their structure or reimagining them as adaptable, dynamic

interfaces. These walls could sense and respond to the people in the spaces they delimit, enabling applications such as ambient lighting, climate control, and embedded displays.

Walls, often overlooked as simple components of a house, can thus serve as a medium to explore HBI. Integrating [Light-Emitting Diode \(LED\)](#)'s or proximity sensors into wall structures has the potential to create dynamic, seamless, and responsive interactions. This innovation reimagines walls and ensures HBI remains central to the future of smart home design.

Sensor technologies, such as motion detection, proximity, and temperature sensors, are essential for capturing human input or presence. For instance, proximity sensors enable touchless control of lighting or HVAC systems, enhancing convenience and hygiene. Similarly, communication protocols such as [Message Queuing Telemetry Transport \(MQTT\)](#) and Wi-Fi, along with real-time data exchange, provide IoT devices with fast connectivity, creating seamless and responsive environments. Equally important are actuation technologies, like LED's, which convert user input into dynamic outputs to make systems truly responsive. Furthermore, advancements in machine learning algorithms are transforming these systems, enabling them to interpret human behavior and preferences. This innovation enhances user interaction and improves the adaptability of smart devices, further advancing smart home technology.

Although HBI systems vary in application, purpose, and technology, the effectiveness of interaction is crucial. Intuitiveness is key, as maintaining simplicity in usability ensures systems accommodate users with varying levels of technical expertise, avoiding overcomplexity is especially important with technologies like gesture recognition to ensure systems achieve their intended purpose.

Integrating technology into daily life also brings privacy risks. Sensors in intimate spaces (e.g., bedrooms and living rooms) may cause discomfort, making it essential to design applications with strategies that prioritize data security and build user trust.

Lastly, balancing advanced functionality with budget constraints is a critical consideration. Systems should adapt to existing third-party setups and avoid unnecessary over-engineering to remain practical and cost-effective.

1.1.3 Interactive Partitions in the Context of Home Automation

Transforming partition walls into interactive surfaces presents significant potential through various innovative applications, aligning with the ongoing shift toward fully automated home environments. While walls have traditionally served as static architectural features, limited to acoustic and insulation improvements, their transformation into dynamic, responsive surfaces addresses the overdue need for innovation in this fundamental aspect of home design.

In response to these limitations, new concepts like “smart walls” are emerging. Featuring touch interfaces and gesture recognition, these walls demonstrate the potential for interactivity but remain costly—both in

terms of energy consumption and financial investment. Moreover, current implementations are primarily directed toward commercial and public spaces, leaving private, intimate home environments underexplored.

Among the available technologies, LED-based implementations offer one of the most promising solutions for introducing interactivity into home walls. They provide an affordable, scalable option with clear visual outputs, enabling endless light effects to adjust a room's ambiance, deliver visual feedback (such as notifications or time displays), and much more. Compared to other advanced technologies, LED's are easier to implement, making them highly practical for residential applications.

While LED's enable effective visual outputs, the interactivity of such systems also relies on sensors to deliver real-time responsiveness. Proximity detection, for example, stands out as a vital technology, particularly in a post-pandemic world where hygiene remains a major concern. This technology allows for contactless control and gesture recognition, enhancing convenience and safety. Additionally, other sensor technologies—such as temperature, sound, and motion sensors—add further value by enabling a wide range of responsive functionalities.

By integrating these systems into home wall partitions, the user experience can be significantly enhanced. These interactive partitions can adapt to various inputs, creating personalized and highly adaptive spaces while improving energy efficiency by automatically dimming or turning off lights based on occupancy. Furthermore, this innovation promotes inclusivity. For instance, individuals with reduced mobility can seamlessly control their environment using interactive walls, offering non-intrusive and accessible solutions tailored to their needs.

1.2 Project Evolution and Current Scope

This dissertation originated within the scope of a larger project focused on the development of an LED-based partition wall intended for domestic environments. The core idea was to create a structural element—specifically, a wall—that would not only fulfill its traditional architectural function but also serve as an interactive surface capable of delivering ambient feedback through dynamic light patterns. This concept aligns with broader trends in ambient intelligence and human-building interaction, where architectural elements become active participants in a smart home ecosystem.

Initially, the project concentrated on prototyping the LED wall itself. This included the creation of a hardware infrastructure controlled by a Raspberry Pi running a Flask server, and the development of a web-based Graphical User Interface (GUI) to allow users to interact with the LED wall in real time. Furthermore, an algorithm was implemented to convert MP4 video content into dynamic LED effects, enabling a rich and personalized lighting experience. These components together formed a robust, if relatively centralized, interactive display system.

However, as the project evolved, it became clear that the original concept of the wall as the primary interactive surface could be enhanced and generalized by introducing sensor-based control mechanisms. This realization led to a significant conceptual and technical shift: the focus moved from the LED wall itself to the development of a **standalone gesture-based control module**. The aim was to create a compact, modular unit capable of recognizing hand gestures via capacitive sensing, with the ability to control not only the LED wall but also a variety of other smart home devices.

This new direction brings several advantages. First, it enables the decoupling of the interaction mechanism (gesture input) from a specific output device (the LED wall), making the solution more flexible and extensible. Second, it allows the gesture control module to be designed as a **product-like unit**, conforming to standard electrical switch dimensions and capable of wall integration. This approach not only broadens the system's applicability but also moves the project closer to a commercially viable solution.

In order to ensure robustness and reliability in gesture recognition—particularly in free-space or noisy environments—machine learning techniques were introduced into the capacitive sensing pipeline. These techniques allow for more sophisticated and adaptive interpretation of raw sensor data, improving classification accuracy and enhancing user experience.

Due to this significant reorientation, the dissertation necessarily includes work that might appear, at first glance, tangential to the gesture module itself. For example, the initial development of the LED wall, its web-based control interface, and the video-to-LED effects algorithm are still documented as they represent both the technological context and a practical use case for the gesture system. Rather than being seen as divergent efforts, these components should be understood as milestones in the project's progression—demonstrating a trajectory from concept exploration to system modularization and user interaction design.

The current scope of the dissertation, therefore, encompasses:

- The design and implementation of a **gesture-based capacitive sensing module**;
- The integration of **machine learning** to enhance gesture recognition robustness;
- The adaptation of the module into a **standard-sized, wall-compatible** device;
- And the documentation of the **initial LED wall prototype** as a proof-of-concept application of the gesture module.

Together, these components form a coherent approach to creating modular, intelligent interfaces for domestic environments—interfaces that bridge the physical and digital layers of the smart home.

1.3 Motivation

The motivation behind this research is deeply rooted in addressing the evolving needs of contemporary living, where the intersection of technology and architecture presents a promising frontier. The surge in the adoption of smart home technologies suggests an increasing market demand for innovative solutions that seamlessly blend functionality and aesthetics.

Interactive partition walls offer a unique opportunity to meet this demand, providing a canvas for interactive functionalities that enhance user comfort, contribute to health and well-being, and enable personalized spatial experiences. By infusing intelligence into these architectural elements there can be a transformation of home partitions into active contributors to a smart, adaptive, and user-centric living environment. The motivation stems from recognition that the spaces we inhabit should not only cater to our basic needs but also actively adapt to our preferences and contribute to an overall sense of well-being.

1.4 Objectives

The overarching objective of this dissertation is to explore how domestic partition walls can transition from passive architectural elements into active, interactive interfaces, while preserving the visual and spatial qualities of the built environment. To this end, the work focuses on the design, implementation, and validation of a gesture-based interaction module that can be unobtrusively integrated into residential settings and used to control different types of actuators.

More concretely, the dissertation pursues the following specific objectives:

- **Design a wall-integrated gesture recognition module (GRM).** Conceive and design a compact hardware module based on capacitive proximity sensing, dimensioned to fit within standard wall switch form factors and suitable for integration behind or within partition surfaces.
- **Develop a robust signal processing and embedded machine learning pipeline.** Implement a data acquisition and pre-processing chain for capacitive signals, and design a lightweight machine learning model capable of performing real-time gesture classification on a resource-constrained microcontroller platform.
- **Integrate the GRM into a modular smart home architecture.** Define and implement the communication and interaction mechanisms that allow the gesture module to operate as a generic input device, decoupled from specific actuators and compatible with different types of domestic systems.

- **Demonstrate and evaluate the system through a concrete use case.** Integrate the GRM with an LED-based partition wall and assess the end-to-end performance of the system in terms of recognition accuracy, responsiveness, and practical usability, while respecting privacy and avoiding the use of cameras or wearable devices.

These objectives collectively aim to demonstrate the feasibility of embedding non-contact, gesture-based interaction directly into architectural elements, leveraging capacitive sensing and embedded machine learning to enable modular and privacy-preserving control of residential environments.

1.5 Document Structure

Following the introductory chapter, the document will be structured into four sections. The second section will provide an in-depth overview of the current state of relevant concepts, offering a technical background to the problem at hand by elaborating on associated technologies and functionalities.

Moving forward, the third section will conduct a careful analysis of the problem and present a detailed description of the solution's design. This section aims to offer a clear understanding of the intricacies of the identified problem and the proposed design solution.

The subsequent section, the fourth, will cover the implementation of the solutions outlined earlier. Post-implementation, a thorough documentation of the testing and verification process will ensure the robustness and effectiveness of the proposed solutions.

Concluding the document in the fifth section, the discussion will center around the results of the testing, drawing comparisons to the initial expectations. Potential avenues for future work may also be explored, contributing to the ongoing discourse on the intersection of technology and architectural design.

Chapter 2

Literature Review

2.1 Smart Environments and Ambient Intelligence

The integration of a wide variety of devices into our living spaces has been a prominent topic for at least the past 20 years. This phenomenon can be described through several overlapping concepts, each emphasizing different academic perspectives. Among them, [Aml](#) (Ambient Intelligence) stands out as a broad and unifying framework. To understand it fully, one must first consider the foundational ideas that preceded and shaped it.

2.1.1 From Ubiquitous Computing to Ambient Intelligence

Rooted in Norman's idea of the "Invisible Computer" and the concept of Ubiquitous Computing, [Aml](#) is a multidisciplinary paradigm that emphasizes anthropocentric models of human-machine interaction, where technology is designed to recede into the background of human perception [1].

The goal of Ubiquitous Computing is defined as the unobtrusive presence of computing throughout the physical environment—computers that are effectively invisible to the user [2]. From this foundation, the concept of Ambient Intelligence emerges as a natural evolution.

This vision offers essential criteria for designing systems in which electronic devices are so seamlessly integrated that they become imperceptible to users, while simultaneously enhancing the range and intuitiveness of human-machine interaction [1]. Such systems are typically characterized by responsiveness, adaptivity, and context-awareness, all of which contribute to the seamless experiences envisioned by Ambient Intelligence.

Realizing these characteristics requires meeting specific technological demands. Consequently, the implementation of modular, low-power devices, along with distributed, high-bandwidth, and heterogeneous networks of sensors and actuators, is critical to enabling such systems [1].

2.1.2 Characteristics of Smart Environments

Smart environments are physical spaces enriched and enhanced through the application of technology, enabling them to be aware of, responsive to, and adaptive towards their occupants, as defined in [3].

These environments can also be characterized by the presence of the following features:

- **Context-aware** computing allows systems to dynamically adapt to user needs based on real-time contextual information. It refers to a system's capacity to detect, interpret, and respond to data such as location, time, or user activity [4].¹
- **Personalization and Adaptivity** denote the ability to tailor user experiences to individual behaviours and preferences. Systems customize interactions and evolve over time, predicting needs and adjusting accordingly [3].
- **Autonomy** is achieved through active context-awareness, allowing systems to monitor real-world conditions and act autonomously without user intervention (e.g., adjusting home heating or alerting emergency services) [4].

Together, these characteristics foster **Natural Interaction**, by leveraging context to seamlessly tailor the experience to each user and integrating technology subtly into the environment so as not to disrupt the user experience.

2.1.3 Human-Building Interaction and Natural Interfaces

As defined in [5], **HBI** is a field focused on enabling built environments that can evolve, learn, and adapt to improve the quality of life of their users while optimizing resource usage and service availability. Furthermore, it studies how built environments can affect human outcomes, as well as how humans interact with, adapt to, and influence the built environment and its systems.

HBI is defined in a broader, more philosophical manner in [6], described as a conceptual shift towards spatiotemporally immersive environments, where interactions unfold over time and space rather than in discrete moments. Additionally, it is seen as a concept that intertwines physical, spatial, and social dimensions, in which buildings are conceptualized not only as physical structures but as interfaces capable of mediating interactions with varied purposes. Lastly, it centers on a human-centric vision, emphasizing human experience and aiming to design environments that adapt to their users, rather than requiring users to adapt to them.

¹In the field of context-aware computing, the term “*awareness*” is used metaphorically; it does not imply that these systems are conscious or sentient.

Within this field, the concept of **natural interfaces** becomes critical (originally a main concept of [HCI](#)), allowing people to interact in a more intuitive manner with these systems. In [\[7\]](#) this concept is described as the capability of a system to use implicit forms of inputs that support natural human forms of communication, including speech, gestures and facial expressions. With the final objective of allowing humans to interact with computer systems in the same way human to human interaction occurs.

Within these different types of interactions **gesture-based interactions** are without a doubt a growing trend in smart environments. As explained in [\[8\]](#) gesture based systems provide an intuitive, user-friendly way to establish approach to controlling many kinds of home devices, leveraging intuitive body movements for seamless interaction. Even though this type of system brings about some challenges in its reliability of gesture recognition, the integration of [ML](#) models can greatly minimize this problem

Within gesture recognition systems, **capacitive sensing** has one of the greatest potentials for bringing about a true natural interface. This technology's capability of being installed behind solid objects allows for an unobtrusive application, which is one of the most important aspects of **natural interfaces** [\[9\]](#).

2.2 Gesture-Based Human-Computer Interaction

2.2.1 Gesture Recognition in HCI

As stated in the previous section, gesture recognition systems are a key field in [HCI](#) due to their ability to enable more natural interaction, bypassing the need for explicit devices like keyboards, touchscreens, or buttons. They support intuitive interactions in various applications, from sign language interpretation to home device control, medical assistance, and augmented reality. By recognizing gestures through computer vision or sensors, digital systems gain the capacity to interpret yet another facet of human expression [\[10\]](#).

Gestures can be static (fixed hand postures) or dynamic (trajectories or movements). Within dynamic gestures, there is a further distinction between discrete gestures (isolated) and continuous gestures (flowing in sequence). These categorizations reflect the richness and complexity of human motion, and each type presents unique challenges for recognition, particularly in real-world environments.

Moreover, gesture recognition makes a system sensitive to human activity and expression, aligning seamlessly with the concepts of [Aml](#) and context-aware systems. Its unobtrusive nature is especially valuable in environments where technology recedes into the background, subtly supporting users without drawing attention to itself [\[10\]](#).

2.2.2 Technologies for Gesture Recognition

Gesture recognition can be implemented through various technologies designed to interpret human motion and expression. These technologies are typically divided into contact-based and non-contact-based systems, each with distinct strengths and weaknesses that make them suitable for different applications [11].

- **Contact-Based Systems**

These systems require physical interaction between the user and the sensing device, encompassing technologies such as mechanical sensors, haptic devices, ultrasonic sensors, and multi-touch screens. They offer advantages like higher accuracy, lower computational complexity, and reduced sensitivity to background noise. However, they may also cause user discomfort, limit freedom of movement, and feel less intuitive. Examples include data gloves (e.g., CyberGlove II), accelerometers (e.g., Nintendo Wii Remote), and multi-touch screens (e.g., smartphones).

- **Non-Contact-Based Systems**

These systems detect and interpret gestures without direct contact, often using wireless sensing technologies. Their applications span several approaches:

- **Cameras (Optical Sensors)** are among the most widely used methods, providing a relatively unobtrusive and user-friendly interface. However, they face challenges such as high sensitivity to lighting conditions, complex implementation, and significant privacy concerns [11].
- **Radar Sensors** represent a growing area in gesture recognition, using technologies like Pulsed radar (e.g., Ultra-Wideband Impulse-Radar, UWB-IR) and Continuous-Wave (CW) radar (e.g., Frequency Modulated Continuous Wave, FMCW radar; Single-Frequency CW, SFCW/-Doppler radar). These systems excel in robustness to lighting conditions, privacy protection, and detecting micro-movements with high precision. Nonetheless, they require a clear line of sight between the hand and sensor and tend to be more sensitive to ambient environmental factors than wearable solutions [12].
- **Capacitive Sensors** operate by detecting variations in surrounding electric fields, which enables versatile implementations for both contact and contactless sensing. They are cost-effective, energy-efficient, and offer low latency, supporting intuitive interfaces suitable for IoT, Virtual Reality (VR), Augmented Reality (AR), touchless interfaces, Human Machine Interface (HMI), robotics, and HCI. Additionally, they can sense through solid materials, achieving truly unobtrusive integration. However, their limitations include high sensitivity to peripheral objects, significant data processing requirements, and limited detection distance [13].

2.3 Capacitive Sensing for Gesture Interfaces

2.3.1 Principles of Capacitive Sensing

In Capacitive Sensing is used to estimate physical properties like touch, proximity, or deformation by measuring the capacitance(i.e., the ability to store charge in an [Electric Field \(EF\)](#)) between two or more conductors. The conductors (often called electrodes) are mainly solid metal parts, but can also be made from other conductive materials, including foils, transparent films, plastics, rubbers, textiles, inks and paints, as well as the human body or objects in the environment. Most capacitive systems measure changes in capacitance coupling between the human body and one or more electrodes.

The role of ground is also important, as it refers to a common potential to which all objects in the system are electrically coupled. This ground serves as a common reference which is critical to operation, it may refer to the electric potential of the floor, the human body itself or the ground plane in a circuit board.

A capacitance exists when two electrodes are separated and are at different electric potentials, if so an [EF](#) exists between them. Capacitance C is the measure of the Q (i.e., quantity of electrons) that a capacitor holds when a certain electric potential V is applied, the equation 1 shows this relation:

$$C = \frac{Q}{V} \quad (1)$$

The SI unit of capacitance is the farad (F). Typical capacitances in [HCI](#) applications are on the order of picofarads (pF or 10^{-12}) and primarily depend on three properties: Size and shape of electrodes, distance between them, and the dielectric properties of any material that lies between them [14].

[15] defines dielectric material as an electrical insulator, which means it resists the flow of electrical current. The dielectric constant (ϵ), or relative permittivity, is a key measure of this effect. It's a dimensionless number representing a material's ability to store electrical energy when exposed to an electric field (as seen in table 1), compared to a vacuum (which has a dielectric constant of 1.0). All materials have a dielectric constant greater than 1. It is calculated using the equation 2:

$$\epsilon = \frac{C}{C_0} \quad (2)$$

Where:

- ϵ is the dielectric constant
- C is the capacitance of a capacitor with the material as the dielectric
- C_0 is the capacitance of the same capacitor with a vacuum as the dielectric

Table 1: Dielectric Constants of Common Materials

Material	Dielectric Constant
Air	1
Alcohol	16–31
Drywall	1.4–2.9
Paper	2.3
PVC	3
Plexiglass	3.2
Silicon	11–12
Wood	2–6
Water at 20°C	80.4
Water at 50°C	78.5

Commonly recognized are two closely related notions of capacitance: self capacitance and mutual capacitance.

- **Self-Capacitance** measures the change in capacitance of a single electrode relative to ground, when a user touches the area surrounding the electrode, additional capacitance is added. In this type of measurement the electrodes project electric field lines 360 degrees, allowing interaction from both side unless shields are used. This wide field projection can limit how closely buttons can be packed together to prevent accidental triggers as well as interfering with each other. Additionally, parasitic capacitances to ground can desensitize the touch, making the change harder to measure. This approach is best suited for proximity gesture recognition applications with a limited number of electrodes [16].
- **Mutual Capacitance** measures the change in capacitance between two distinct electrodes: a transmit (Tx) and a receive (Rx) electrode. When a user interferes with the area between the two electrodes, the mutual capacitance between them is reduced because the user's body disturbs the electric field propagation. The electric field is tightly coupled between the two conductors, enabling higher-density pairs of electrodes without cross-coupling issues, furthermore parasitic capacitances to ground do not affect the measurement in this systems, though they can impact the ability to drive electrodes at high speeds. Mutual capacitance is well-suited for button matrices and advanced sensors with small node sizes and high electrode density [16].

Figure 1 illustrates this difference.

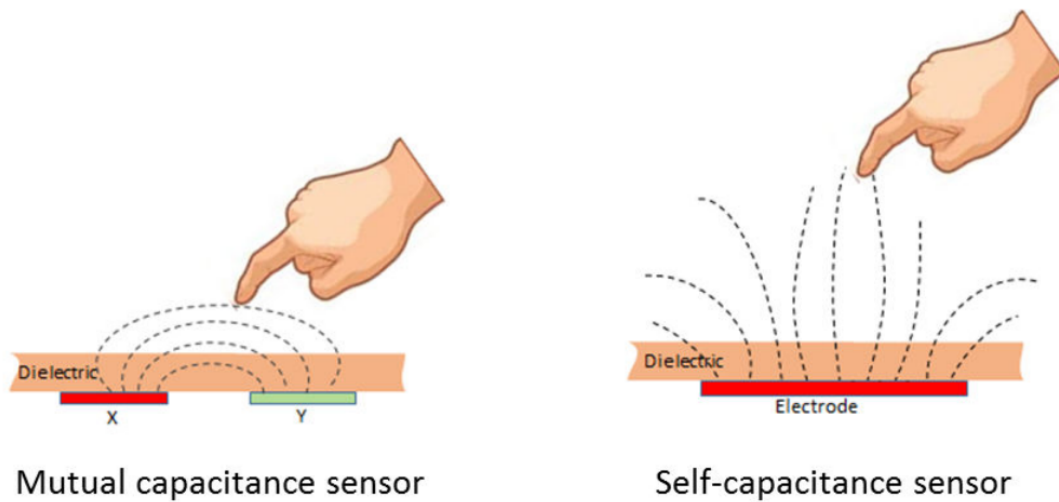


Figure 1: Self vs Mutual Capacitance

Capacitive sensing, especially when applied to gesture recognition interfaces in [Aml](#) scenarios, offers several **key advantages**:

- **Low power consumption:** Ideal for systems that are functioning all the time, in embedded or battery-powered systems [16];
- **Integration flexibility:** Electrodes can be printed, milled, or embedded behind materials like wood, plastic, plaster or glass, preserving aesthetics and enabling true unobtrusive interfaces [14];
- **High sensitivity:** Enables detection of slight movements, helping in gesture recognition applications;
- **Fast response time:** Data of variation in capacitance can be measured and acquired almost instantly, supporting real-time interaction;
- **No moving parts:** Enables sturdy mechanical systems with high reliability.

However, several **limitations** must be considered:

- **Sensitivity to environmental conditions:** Humidity, temperature, and nearby objects can cause interference, introduce drift and noise in measurements [9];
- **Parasitic capacitances:** Especially in self-capacitance systems, unintended capacitive paths can reduce measurement accuracy;
- **Calibration requirements:** Periodic recalibration may be necessary to maintain reliable detection over time.

These trade-offs underline why capacitive sensing is often chosen for simple, robust proximity or coarse gesture recognition in ambient intelligence, rather than fine-grained gesture tracking. The choice between self-capacitance and mutual capacitance further depends on design goals such as required spatial resolution, electrode density, and desired application.

2.3.2 Capacitive Sensor Design

Sensor design is foundational to capacitive gesture interfaces, enabling a move towards more natural, non-contact human-computer interaction (HCI)[17]. Unlike traditional input devices, capacitive sensors can be hidden behind materials like wood or plastic without losing functionality, making them ideal for unobtrusive applications in smart environments and ambient intelligence scenarios [9]. The basic principle involves measuring the variation of the electric field (EF) caused by a nearby conductive object, such as a human hand [13].

2.3.2.1 Design Principles and Challenges

As seen in the previous section capacitive sensing relies on measuring small changes in capacitance caused by the proximity of conductive objects, such as human hands. The key objectives in designing a system include being able to achieve high sensitivity, low noise susceptibility and maintaining stable performance despite changes in surrounding environment. In order to do that, the **principles** that design has to follow are the following:

- **Non-Contact Operation:** Systems are designed to measure capacitive variation on the electric field, triggered by hand gestures near the sensor without physical contact [13];
- **Low Cost and Power:** A key part of this systems is the use of cost-effective and low-power components, making the technology suitable for commercialization in domains like IoT and HCI [17];
- **Adaptability:** The system should be able to dynamically adapt to inconsistent signal ranges caused by environmental factors [13]

As for the **challenges** that are needed to take into account:

- **Environmental Susceptibility:** Non-contact capacitive sensing is easily affected disturbances from surrounding objects and environmental conditions, which cause unstable signals, noise and drift. This causes distance measures to be relative and not absolute. Therefore, better signal processing is critical [9].

- **Limited Detection Distance:** A fundamental drawback is the limited distance between sensor and target. The system described in [17] has an approximate range of under 15cm.
- **Lower resolution:** Compared to other systems like vision-based ones, capacitive sensing systems have a lower resolution and a higher signal-to-noise ratio, which can make it hard to interpret complex gestures [9].

2.3.2.2 Types of Capacitive Measurement Techniques

When discussing different measuring techniques for this technology, there are mainly two mechanisms:

- **Contact Sensing:** This technique requires direct physical contact with the sensing surfaces. It is the highly used in commercial applications like touch sensing, localization, existence detection and contact sensing applications in the field of HCI. Most systems that use this method for gesture recognition require the use of wearable devices or recognition of static gestures [13].
- **Non-Contact (proximity) Sensing:** This is the primary focus of most recent studies as it shows the most potential for innovation. As studied in [13] it measures the electric field capacitance variation around a sensor caused by nearby objects without direct touch. This method allow sensing through solid objects with low conductivity.

2.3.2.3 Electrode Geometry and Layout

In capacitive sensing, the sensor's electric field extends out from the sensing surface. Therefore the **size and shape** of that surface are a determining factor.

As far as the electrode's **size** there is a golden rule: The larger the area of the electrode the better the range, this can be explained easily when analysing Equation 3:

$$C = \frac{\epsilon_r \epsilon_0 A}{d} \quad (3)$$

Where we can see the capacitance value depends directly of the value of the area (A).

This can be further demonstrated with testing as seen in the figure 2. There is however the trade-off of the sensitivity to outside noise increases in proportion [18].

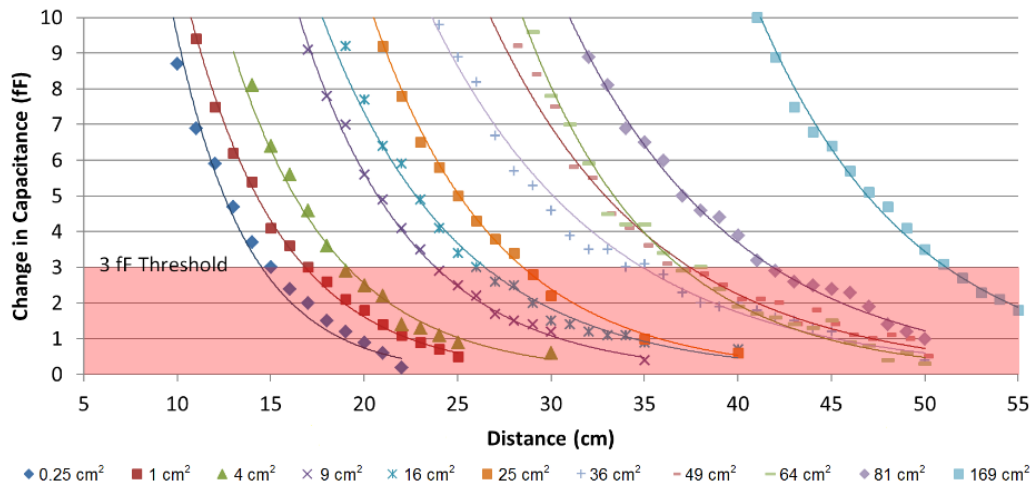


Figure 2: Capacitance vs Distance for Various Sensor Area Sizes

The **shape** of the electrode is critical as it will consequently shape the electric field that will surround it. Most popular applications use one of the following shapes:

- **Rectangular Strips:** This shape is ideal for creating a directional sensor. A rectangular strip has a higher perimeter-to-area ratio compared to a square of the same area, which can lead to higher sensitivity along its longer edges. This property is useful for applications like linear sliders (Figure 3), liquid level sensing, or detecting the direction of an approaching object.



Figure 3: Linear Slide Electrode

- **Squares & Circles:** These geometries are used when a symmetrical, non-directional sensing field is required. They are sensitive to approaches from any direction in the plane. A circle has the minimum possible perimeter for a given area, which can result in a more concentrated and uniform electric field compared to other shapes. Squares are extremely common due to their ease of design and manufacturing on printed circuit boards (PCBs).

- **Interdigitated Electrodes:** One of the most common and heavily studied electrode shape, it consists of two interlocking, comb-like conductive structure. It is designed to maximize the interaction of the fringing electric field with a material placed near its surface (See Figure 4). Because of their high surface sensitivity, they are used widely for chemical and biological sensors, material characterization, humidity sensing, and measuring dielectric properties of liquids [19].



Figure 4: Interdigitated Electrodes

- **Fractal Electrodes:** More advanced research has explored the use of fractal geometries to enhance sensor performance. Fractal shapes, such as Hilbert curves or Sierpinsky carpets are used to maximize the perimeter in a given area. A longer perimeter strengthens the fringing electrical field and improves sensitivity compared to simple euclidean shapes [20].

The layout of the electrodes is usually very heavily dependent on the application. Nevertheless there are some configurations which are widely used for gesture recognition systems:

- **Directional Arrays:** A common layout for gesture recognition where a limited number of electrodes are used to detect gesture along specific axes. In [17] a cross configuration is used to gestures like swipes in the four directions.
- **Matrix or Grids:** A more complete version of the directional array, by covering all coordinates a given area, the amount of gestures that can be recognized increases drastically. It can be used for capacitive touchpads, high resolution proximity sensing, and recognizing complex gestures like drawing shapes [14].
- **Circumferential Arrays:** In this layout the electrodes are positioned in around the perimeter of an object, such as a user's wrist. This application is adequate for wearables as it can sense the orientation or gestures like twisting and turning [21].

- **Linear Arrays:** A simpler configuration where the electrodes are arranged in a single line. Used in applications where there is the need to detect the position and movement of objects along a single dimension. Commonly used in linear slides or liquid level sensing.

2.3.3 Real-World Applications

MAY BE UNNECESSARY

2.4 Machine Learning for Gesture Recognition

2.4.1 Motivation for ML in Capacitive Sensing

The implementation of ML stems from the huge improvement it offers to systems that traditionally present some problems and challenges in the processing and interpretation of raw sensor data. Below are presented some challenges that appear when trying to interpret sensor signals.

Non-Linearity refers to sensor outputs that are not directly proportional to the input, and signals whose statistical properties change overtime (non-stationary). This often requires advanced processing to interpret. Some examples of this phenomenon are the electrical activity of muscles, especially during movements where muscle length and joint angles change, the mathematical models for these signals are difficult to apply in real-time because of the challenges in estimating their parameters [22]; The design of the data acquisition hardware itself can introduce non-linearity. One custom circuit for converting capacitance to voltage was noted to create a non-linear relationship between the input capacitance and the final output voltage [23]. For this specific challenge there has been proven benefits in implementing ML classifiers like Multi-Layer Perceptrons for their ability to model complex, non-linear relationships between sensor inputs and gesture outputs [24].

Signal Noise and Interference presents a notable disturbance to the interpretation impacting the clarity of the sensors signal. One of the most difficult aspects of this interference is the large number of possible sources, such as the environmental [Electromagnetic Interference \(EMI\)](#) [24], [Alternating Current \(AC\)](#) frequency of power lines [22] and many other specific possible sources. In addition to the most common ways of fighting this issue, such as active hardware shielding and signal filtering, ML offers critical help in "ignoring" random variations and noise focusing on extracting the most important patterns in a signals's signature. Specifically [Convolutional Neural Networks \(CNNs\)](#) are used as they can function as not only classifiers but also powerful feature extractors [23]

Multi-User variability is also a major challenge in gesture recognition as signals can vary widely from person to person due to anatomy, performance of the gesture and even ethnic factors like skin tone. Therefore the diversity and number of people used in the training dataset of recognition systems are two critical aspects in the performance of the recognition system [25]. In [22] this aspect is clearly explained, differentiating between individual and general models that work for any user who was not part of the training data.

ML-based classifiers are currently the dominant methodology in the field due to their undeniable contributions in improving robustness, adaptability, and higher detection accuracy as verified in [17] where an accuracy of 99.37% was achieved. Reviews of gesture recognition systems present mostly ML and deep learning algorithms, including Support Vector Machines (SVMs), Artificial Neural Networks (ANNs), Decision Trees, Naïve Bayes, Hidden Markov Models (HMMs), CNNs, and Recurrent Neural Networks (RNNs), demonstrating their importance in the technology [25] [22] [24].

2.4.2 Learning-by-Example Paradigms

The learning-by-example approach to gesture recognition is a commonly used method that is based on providing multiple, and diverse demonstrations of a gesture where the system averages these examples and creates a general template for said gesture [9].

Recent studies on capacitive sensing demonstrate how learning-by-example paradigms (i.e., data-driven model training on labeled gesture samples) can enable real-time gesture recognition with high accuracy. The research done by [13] and [17] implements a pipeline structure where the raw electrical field signal is sent to a preprocessing that aims to improve signal quality and isolate gesture segments before classification. Before being fed into learning algorithms there is still an adaptive thresholding dynamically detecting gesture onset and offset which enables accurate frame extraction. Finally, the learning algorithms are able to learn gesture patterns from examples without the need for handcrafted features.

In this same research [17] there is an evaluation of 18 different classifiers for this application. The models evaluated include traditional boosting-based machine learning, CNNs, RNNs, and Graph Neural Networks (GNNs) deep learning models. The results were as follows:

- A total of 18 candidate models were evaluated across four main paradigms: CNNs-based, RNNs-based, GNNs-based, and boosting-based machine learning models, all trained on the same framed capacitive gesture dataset.
- RNN-based models using a GRU architecture achieved the highest overall classification accuracy (approximately 99.37%) with stable training dynamics and low computational cost, making them well-suited for real-time gesture recognition.

- **CNNs** models performed poorly on raw time-series data (accuracy below 50%), as they could not extract meaningful temporal patterns directly. After applying Continuous Wavelet Transform pre-processing, their accuracy improved to 92.36%, but this came at the cost of high computational requirements, making them less suitable for real-time deployment.
- **GNNs** models reached 96.18% accuracy by leveraging spatial relationships between sensors. However, their generalization was limited for small sensor arrays and their structure required redesign if the sensor topology changed, reducing their practicality for flexible systems.
- Boosting-based machine learning models exhibited the lowest performance, with accuracies ranging between 30–53%, and were primarily used as baselines.
- Considering accuracy, training behavior, and computational efficiency, the GRU-based RNN model was ultimately selected as the final classifier for integration into the real-time interface control pipeline.
- This choice aligns with earlier work that also employed a GRU classifier to achieve high accuracy on multiple gesture classes.

The combination of robust framing, effective preprocessing, and deep learning architectures demonstrates the power of learning-by-example approaches for low-latency, high-accuracy gesture recognition, and may represent a key turning point in the broader implementation of this technological application.

2.4.3 Embedded and Edge Deployment

Deploying gesture recognition models on embedded and edge devices involves operating within tight resource constraints, such as limited memory, processing power, and energy availability. These conditions require careful consideration of model size, inference latency, and accuracy, alongside the selection of suitable deployment frameworks and hardware platforms. Recent developments in lightweight neural networks and edge toolchains have made it increasingly feasible to run capacitive sensing models directly on microcontrollers, enabling responsive and efficient on-device inference.

2.4.3.1 Constraints for Embedded ML

The core constraints to evaluate when planning the deployment of embedded **ML** systems include model size, **Random Access Memory (RAM)** and flash memory footprint, inference latency, supported operations, power budget, as well as toolchain support and ease of deployment. Consequently, several real-world trade-offs must be considered. For example, shrinking models and moving inference on-device often reduces

power consumption and mitigates privacy risks, but may increase latency if the MCU is too resource-constrained. Similarly, [RAM](#) and flash limitations impose restrictions on model selection and the size of intermediate buffers [26].

Within these constraints, practical capacitive sensing systems must balance recognition accuracy with footprint and real-time responsiveness.

2.4.3.2 Practical Examples

In [27] a textile-integrated capacitive sensing solution was implemented, this article is a clear demonstration of capacitive signals feeding embedded [ML](#) pipelines. Furthermore, in [28] a ring based capacitive interface was developed using gesture recognition for multi-finger tracking, it demonstrates the use of [Long Short-Term Memory \(LSTM\)](#)-based capacitive sensing on embedded targets. Finally, [21] provides a good example of capacitive sensing with on-board inference (TinyML), using four electrodes attached to a wrist-band this application implemented a 1-D [Convolutional Neural Network \(CNN\)](#) deployed with TensorFlow Lite on a Cortex-M4. Showing promising values and achieving 96.4% accuracy with a 29.6 KB model, on-board 26.4mW power consumption and 12ms inference. Showing undeniable proof that capacitive solutions with embedded [ML](#) is viable at the edge.

2.4.3.3 Edge Deployment Frameworks and Toolchains

Deploying gesture recognition models on microcontrollers requires efficient toolchains capable of converting trained models into highly optimized code for low-power, memory-constrained hardware. Frameworks such as TensorFlow Lite RT, CMSIS-NN, and Edge Impulse provide the necessary infrastructure to bridge model development and embedded deployment, enabling capacitive sensing applications to run inference directly on devices like the ESP32 or Cortex-M microcontrollers.

- **TensorFlow Lite RT (formerly TensorFlow Lite Micro)** is Google's unified runtime for deploying machine learning models across a wide range of edge devices, including bare-metal microcontrollers. It replaces the earlier TensorFlow Lite Micro framework without altering its original goals, maintaining a design that uses no dynamic memory allocation, relies entirely on static arena buffers, and keeps the binary size minimal. It supports 8-bit quantized models for deployment under strict memory and power constraints, and works across various MCUs including ESP32 and Arm Cortex-M processors [29].
- **CMSIS-NN** is a software library offering a collection of neural network kernels optimized to maximize performance and minimize the memory footprint of neural networks on Arm Cortex-M processors [30]. It provides significant benefits in terms of latency and memory efficiency, particularly for Conv1D and dense layers [31].

- **Edge Impulse** is an end-to-end platform for creating custom datasets, developing models, and deploying them on edge devices. It includes built-in signal processing and model optimization capabilities, offering broad platform support, ease of use, and automation [32].

These frameworks offer complementary approaches for deploying capacitive gesture recognition models on embedded platforms. TensorFlow Lite RT provides a robust and portable runtime for quantized neural networks, suitable for both ESP32 and Cortex-M devices. On Arm platforms, CMSIS-NN can be optionally integrated to optimize convolutional and dense layers for reduced latency and energy consumption. Edge Impulse simplifies data collection and model prototyping, which is particularly valuable during the early stages of development, although it is not required for final deployment. Ultimately, the choice of toolchain depends on the target hardware and project goals: TensorFlow Lite RT alone may suffice for many applications, while CMSIS-NN or Edge Impulse can be selectively incorporated to enhance performance or accelerate development.

2.5 Synthesis and Research Gap

2.5.1 Synthesis of Reviewed Literature

The literature reviewed previously collectively points towards a growing convergence between technological innovation and human-centered design, where the surrounding environment itself becomes sensitive to and active in human experience. Research across domains such as ubiquitous computing, [HCI](#) and [IoT](#) systems reinforces this clear movement from isolated smart devices into **integrated smart environments**. These environments make use of networks of sensors, actuators, and intelligent algorithms to analyse user behaviour, interpret contextual information, and adapt dynamically to individual needs.

In the core of this evolution is the concept of [Aml](#)- a paradigm that envisions environments that are sensitive to their inhabitants behaviours, as well as being capable to respond and adapt in unobtrusive ways, creating a layer of abstraction between the user and currently employed interfaces. According to Braun in [9], [Aml](#) represents a foundational vision of invisible, context-aware interfaces seamlessly integrated within ordinary spaces and objects. Rather than relying on specific user commands or overt interfaces, such systems anticipate user's intentions through contextual awareness and subtle interaction modalities.

Within this context, **gesture-based interaction** has emerged as a particularly compelling modality in [HCI](#). Comparing with more traditional input methods, gesture interaction allows natural, intuitive communication that reduces cognitive load and preserves user's attention. Furthermore, it represents a solid bridge between physical and digital realms, enabling fluid exchanges that align with the [Aml](#) core idea of unobtrusive technology.

Among the various technologies available enabling such interactions, capacitive sensing presents a unique blend of advantages being: low cost, minimal power consumption, ability to be "hidden" behind non-conductive materials. However, studies and technical documents highlight that capacitive sensing are heavily constrained by electrical noise, limited sensitivity, and signal resolutions. In [17] there is clear documentation that further elucidates how material properties, grounding effects, and environmental interference can distort sensor readings, complicating the reliable extraction of gesture signatures in practical applications.

To address these limitations, recent research increasingly integrates ML approaches that can tackle the low reliability and non-linearity of capacitive signals. Since [9] early learning-by-example framework, contemporary works now apply, advanced models, such as CNNs, SVMs, and LSTM networks, to recognize complex gesture patterns from noisy capacitive or radar-based sensor data. These methods increase robustness and significantly open the range of possible application of this type of technology.

In summary, existing research has extensively explored the individual building blocks of intelligent environments—gesture recognition using capacitive sensing, and ML deployment. Yet, there remains a notable gap in integrating these elements into a unified, compact, and intelligent wall-mounted interface that embodies the Aml principle of invisible interaction. This gap defines the rationale for the present project, which seeks to synthesize these parallel lines of innovation into a cohesive and deployable smart-environment interface.

2.5.2 Identified Gaps in Current Research and Practice

Existing literature reveals several persistent gaps that hinder the realization of holistic, deployable smart-environment interfaces.

Most notably, no integrated system effectively combines capacitive gesture sensing, on-device ML classification, and modular hardware aligned with real-world electrical standards. Current works often examine these components in isolation, resulting in research prototypes rather than scalable solutions.

A pronounced TinyML gap remains: many ML approaches rely on off-board training and cloud inference rather than embedded processing. Early projects such as MoCaPose and PeriSense demonstrate feasibility but remain limited to laboratory-scale prototypes [21].

Furthermore, capacitive gesture systems are largely experimental, not engineered for domestic or retrofit applications. In [9] it is highlighted the lack of robustness and environmental tolerance required for real-world deployment.

Commercial smart devices like Philips Hue Tap and Aqara wall switches illustrate modularity and integration within smart ecosystems but still depend on touch or voice, with gesture input largely unexplored.

Few studies address environmental robustness, such as humidity, electrical noise, or material interference, despite these being key sources of performance degradation in capacitive sensing. Moreover, the field lacks standardized evaluation protocols, unlike vision-based gesture systems with established benchmarks.

In summary, research has advanced core technologies—sensing, ML, and modular IoT—but not their integration into unified, robust, and deployable interfaces. Addressing this convergence remains essential for realizing adaptive, gesture-based smart environments.

2.5.3 Concluding Remarks

The literature reviewed presents substantial progress in smart environments, capacitive sensing, and embedded machine learning. However, there is also a lack of integration and deployability, most existing solutions remain experimental, fragmented, or unsuitable for real-world implementation.

The following chapters address these gaps through:

- The development of a coherent system architecture integrating sensing, computation and communication layers;
- A hardware-software design that ensures efficient interaction between embedded ML models and a capacitive sensor array;
- Implementation of lightweight learning algorithms optimized for on-device gesture recognition.

In summary, the proposed system seeks to merge the robustness of machine learning, the subtlety of capacitive sensing, and the practicality of modular smart-home design into a single, deployable interface—bridging the gap between laboratory prototypes and real-world applications of gesture-based ambient intelligence.

Chapter 3

Design and Integration of the Gesture Recognition System

This chapter presents the analysis, design, and overall architecture of a capacitive-sensing-based gesture recognition module. Its main focus is to provide an in-depth description of the constraints, requirements, design choices, and technical considerations encountered throughout the development of this system. Both hardware and software decision-making processes will be explained in detail, outlining the rationale behind each component and configuration.

As one of the objectives of the gesture module is to ensure compatibility and seamless integration across different systems, this chapter will also explore the integration strategy adopted to incorporate the proposed gesture recognition module into a representative real-world use case. Only the project-specific segments of the target system will be discussed in detail, with emphasis placed on the interfaces and mechanisms required for successful deployment.

3.1 System Overview

3.1.1 System Purpose and Scope

The developed system aims to enable gesture-based control of smart home appliances and devices (e.g., LED walls, lighting, HVAC units, etc.) through a modular capacitive sensing unit integrated within a local network. The following analysis and design are focused on modularity, as the system is intended to operate when attached to a wide range of devices.

Below is a general illustration of the purpose and scope of the designed module when integrated into a higher-level system. By embedding all sensing and classification logic within the module itself, and outputting only abstract gesture commands, the system remains open to integration with diverse target applications.

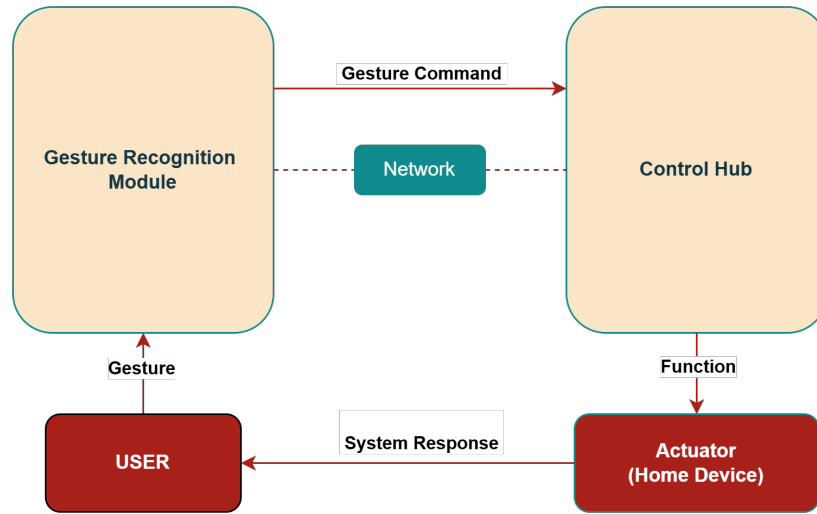


Figure 5: System Overview

Figure 5 illustrates the overall system context, showing how the user interacts with the gesture recognition module and how this module interfaces with the control and actuator layers.

As shown in the diagram, the system is composed of three main subsystems. The next section details their function and interaction:

- **GRM:** As the core component of this project, the gesture recognition module is responsible for capturing user input, processing the data, classifying the gesture, and sending the corresponding command to the control hub.
- **Control Hub:** Represents a home automation controller responsible for managing the behaviour of different actuators (home appliances/devices). Its role in this system is to receive the gesture command from the GRM and translate it into a device-specific action.
- **Actuator:** Represents a specific study case for system operation—for example, a LED wall or window blinds that are controlled through predefined gestures.

Figure 5 also illustrates the internal data flow, which can be divided into six stages corresponding to the system's control and logical transformations:

1. **Sensing Stage (GRM):** Sensor electrodes capture raw capacitive signals, which are then processed by the sensor's internal logic.
2. **Processing Stage (GRM):** The signals are filtered and pre-processed to reduce noise and extract relevant temporal patterns.

3. **Inference Stage (GRM):** A lightweight ML model classifies the gesture based on the processed data, producing a gesture label.
4. **Communication Stage (GRM → Control Hub):** The gesture event is transmitted to the control hub, which receives and interprets the message.
5. **Control Stage (Control Hub):** The control hub maps the gesture label to a configured action and sends the corresponding control signal to the actuator.
6. **Actuation Stage (Actuator):** The actuator receives the command and executes the associated action.

Together, these elements define the complete operational flow of the system, establishing a modular and extensible framework that can be seamlessly adapted to a variety of smart home contexts and device configurations.

3.2 Gesture Recognition Module Design

3.2.1 Requirements Analysis

Before proceeding with the design, it is essential to establish a clear set of requirements guiding the development of the GRM. As the proposed module is intended as a general-purpose interaction interface suitable for integration into a wide range of smart home actuators, the requirements must ensure functionality, robustness, safety, and compatibility with real domestic environments.

Functional Requirements

These requirements specify what the gesture recognition module must *do* to fulfill its role as a versatile and modular smart-home interaction component.

- **Detect Hand Presence and Movement:** The module shall detect a user's hand approaching or moving over the sensing surface through capacitive measurements.
- **Recognize a Set of Defined Gestures:** The module shall classify user gestures (e.g., swipe left/right, push forward, hold) using embedded machine learning methods.
- **Generate High-Level Commands:** The module shall translate recognized gestures into abstract control commands that remain independent of any specific actuator.
- **Provide Real-Time Output:** Gesture recognition and command transmission shall occur with minimal latency to ensure smooth interaction.

- **Communicate with External Systems:** The module shall transmit recognized gestures to a controller via a network protocol (e.g., Wi-Fi/[Transmission Control Protocol \(TCP\)](#)), enabling integration with diverse actuators.
- **Support Calibration and Configuration:** The system shall support initial calibration and configuration through software or a user interface.

Non-Functional Requirements

These requirements describe how well the module must perform and the constraints under which it is expected to operate.

- **Compact Form Factor:** The module shall fit within the standard dimensions of a residential wall switch box (aprox. 10x10cm).
- **Low Power Consumption:** The module shall operate efficiently to reduce power usage and enable potential future battery-powered variants.
- **Robustness to Environmental Variations:** Recognition accuracy must remain acceptable under humidity changes, electrical noise, and variations in user behaviour.
- **Real-Time Performance:** Embedded ML inference time shall remain below a defined threshold (e.g., <50 ms).
- **Scalability and Interoperability:** The module shall support integration into broader smart-home ecosystems (e.g., [MQTT](#), [Hypertext Transfer Protocol \(HTTP\) Application Programming Interface \(API\)s](#), Matter).
- **User-Friendly Interaction Experience:** The system should minimise false positives and negatives to ensure consistent gesture recognition.

3.2.1.1 Gesture Set Definition & Interaction Requirements

The gesture recognition module is designed to detect four directional free-hand gestures: **swipe left, swipe right, swipe up, and swipe down**. These gestures were chosen for their simplicity, intuitive use in smart-home interactions, and suitability for modular control scenarios. Their inclusion imposes the following requirements on the sensing system:

- **Two-axis spatial resolution:** the sensing surface must discriminate motion along both horizontal and vertical axes.

- **Distinct temporal activation patterns:** electrode signals should exhibit sequential variations that uniquely indicate gesture direction.
- **Free-space sensing capability:** the system must detect hand proximity above the acrylic (or wall-equivalent) surface without physical contact.
- **Noise robustness:** capacitance measurements must remain stable despite environmental or material-induced disturbances.
- **Adequate gesture sampling:** the acquisition and processing pipeline must sample quickly enough to capture short, dynamic gesture motions for reliable ML classification.

Based on these requirements, there is sufficient foundation to support the module's design, ensuring that the resulting system is both technically sound and feasible for real-world deployment. The following sections build on this analysis to present the design choices and architectural decisions.

3.2.2 System Overview

An initial high-level architecture of the GRM system can be seen in Figure 6, where its composition and how each part interacts with the rest is clearly depicted.

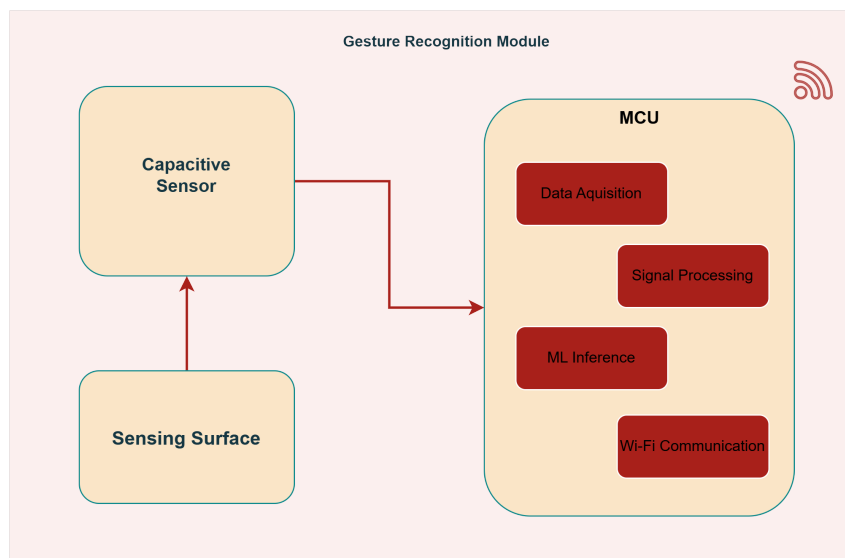


Figure 6: High-level conceptual architecture of the gesture recognition module

There are three clear parts on this system:

- **Sensing Surface:** This component is responsible for the detection of capacitance in electric field surrounding the module, the signals will be the input for the capacitive sensor.

- **Capacitive Sensor:** The sensor that has the function of acquiring the capacitance signal of the electrodes, converting it to a digital signal and communicating with the [Microcontroller \(MCU\)](#);
- **MCU:** The unit responsible for acquiring the digital signal of the capacitive sensor and filtering/processing. Here there will also be deployed a [ML](#) inference process for gesture classification and subsequent communication with the surrounding home automation controller.

3.2.3 Hardware Architecture

This section will be dedicated to the description of the conceptual hardware architecture of the [GRM](#), outlining the design choices and that will guide the subsequent implementation. Although this design is targeted towards a prototype scenario, the base structure of the developed work will be suitable for further development and upgrades.

This section will cover the relevant hardware architecture regarding the three main components presented in [Figure 6](#), developing deeply the decisions and methods to be used in the implementation of this system.

3.2.3.1 Signal Acquisition & Capacitive Measurement IC

The signal acquisition layer is responsible for converting variations in capacitance, induced by the proximity of the user's hand, into stable and high-resolution digital measurements suitable for generating clear gesture signature samples for subsequent [ML](#) classification. This process requires a capacitive measurement system with the following characteristics:

- High sensitivity to small capacitive changes;
- Low-noise measurements;
- Multiple independent sensing channels;
- Sufficient sampling rate (>50 Hz);
- Digital output compatible with embedded [ML](#) processing;
- Efficient interfacing with the processing unit;
- Active shielding capability.

Capacitance-to-Digital Converter (CDC) Options and Selection

To support free-space capacitive gesture recognition, the signal acquisition subsystem relies on a high-resolution capacitance-to-digital converter (CDC). This device transforms small variations in electrode capacitance into stable, digitized measurements suitable for preprocessing and machine learning classification on the microcontroller. Considering the requirements defined earlier—namely multi-channel capability, high sensitivity, robustness to environmental noise, and straightforward digital integration—three CDC families were identified and are discussed below:

1. AD7746 (Analog Devices)

The AD7746 is a high-precision, instrumentation-grade CDC designed for accurate capacitive sensing in precision measurement applications.

Key Characteristics:

- 24-bit, no-missing-codes output with excellent noise performance (effective resolution up to ~ 21 bits, minimum resolution around 4 aF, accuracy of ± 4 fF, and typical linearity of $\pm 0.01\%$).
- Two capacitance input channels configurable as single-ended or combined as one differential channel.
- Conversion rates in the 10 Hz to 90 Hz range, optimized for floating capacitive sensors (grounded-plate variants are addressed by the AD7747).

Pros:

- Very high measurement precision and absolute accuracy.
- Well suited for instrumentation and calibration-grade sensing.

Cons:

- Limited to two channels and relatively low update rates, making it less suitable for dynamic or multi-electrode sensing.
- Higher cost and integration effort compared to multi-channel capacitive controllers intended for UI or gesture sensing.

2. FDC2214 (Texas Instruments)

The FDC2214 is a multi-channel, high-resolution CDC based on a resonant LC sensing architecture, targeting proximity, gesture recognition, and liquid-level measurement applications.

Key Characteristics:

- Up to 28-bit resolution with a system noise floor of approximately 0.3 fF at low output data rates.
- Four differential LC sensor channels (IN0A/B–IN3A/B) using frequency-based conversion, with a maximum output rate of approximately 4.08×10^3 Hz for a single active channel.
- Support for excitation frequencies from 10 kHz to 10 MHz, large input capacitances (up to 250 nF), and an EMI-resistant I²C-based architecture operating at 2.7 V to 3.6 V.

Pros:

- Very high sensitivity and resolution for resonant LC-based proximity and displacement sensing.
- Strong rejection of out-of-band noise and EMI.

Cons:

- Requires external inductors and tuned LC networks, increasing design and layout complexity.
- Higher power consumption and BOM cost compared to grounded-plate capacitive sensing solutions.

3. FDC1004 (Texas Instruments)

The FDC1004 is a high-resolution, four-channel [CDC](#) designed for grounded capacitive sensors in proximity, gesture, level, and material detection applications.

Key Characteristics:

- Four grounded capacitive input channels (CIN1–CIN4), supporting single-ended and differential configurations.
- Effective 16-bit resolution with a full-scale range of ± 15 pF and programmable offset compensation up to 100 pF via an on-chip CAPDAC.
- Integrated active shielding drivers, I²C interface, and operation from a 3.3 V supply with selectable output data rates of 100 Hz, 200 Hz and 400 Hz.

Pros:

- Native support for four grounded channels, well suited to multi-electrode spatial sensing.
- CAPDAC and calibration features simplify baseline compensation with relatively low integration complexity.

Cons:

- Lower effective resolution than resonant LC-based devices, though sufficient for free-space proximity sensing.
- Requires careful electrode layout and calibration to mitigate parasitic and environmental effects.

Considering the requirements defined in Section 3.2.1, the FDC1004 emerges as the most appropriate choice for the proposed gesture module. While the AD7746 offers excellent precision, its limited channel count makes it unsuitable for four-electrode spatial sensing without additional multiplexing. The FDC2214, although capable of gesture recognition, relies on resonant LC architectures that introduce unnecessary design complexity for this prototype. In contrast, the FDC1004 natively supports four grounded inputs, operates directly with planar electrodes, and integrates offset compensation and shielding support, making it well aligned with the needs of free-space capacitive gesture sensing. For these reasons, the **FDC1004** is selected as the CDC for the gesture recognition module.

Detailed Functional Description of FDC1004

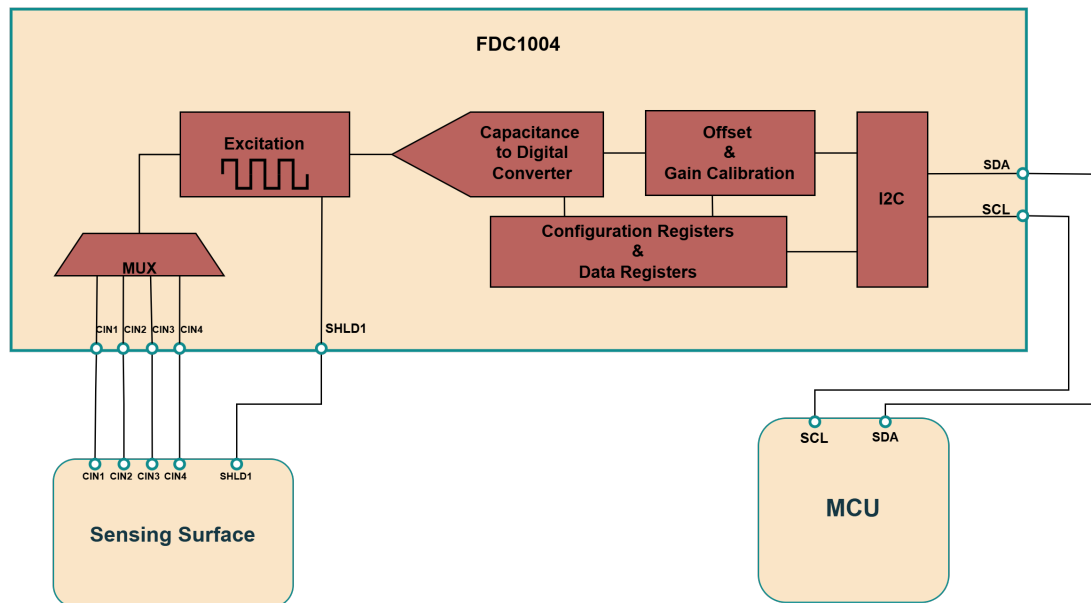


Figure 7: Functional Diagram of the FDC1004, based on the datasheet [33]

Measurement Principle and Channel Usage — Functionally, the FDC1004 periodically excites each input channel and digitizes the resulting capacitance using an internal conversion and filtering pipeline. This provides a clean 16-bit digital measurement per channel that can be transmitted directly to the microcontroller without additional analog circuitry.

The FDC1004 supports two measurement modes: single-ended measurements, which sense capacitance between an electrode and ground, and differential measurements, which sense capacitance between two

electrodes. **Single-ended** operation is selected, as it allows all four channels to be used independently for directional gesture sensing and produces sufficiently strong electrode-to-ground capacitance variations in response to hand motion.

As illustrated in Figure 7, the sensing surface consists of four electrodes, each connected to a single-ended input channel (CIN1–CIN4). Each channel has a full-scale range of ± 15 pF and supports up to 100 pF of offset capacitance. Internally, the **Integrated Circuit (IC)** multiplexes these channels through a shared excitation and conversion path, producing digital values representing capacitance changes above the sensing surface. Given the sensitivity to parasitic coupling and nearby conductive objects, the use of active shielding is critical to maintain measurement stability.

Offset and Gain Configuration — Variations in electrode geometry, parasitic capacitance, and environmental conditions lead to differing baseline capacitance levels. The FDC1004 allows per-channel offset and gain configuration, enabling the effective measurement range to be aligned with the expected gesture-induced variations. This capability reduces environmental impact and facilitates normalization across channels, which is particularly beneficial for gesture classification where relative changes are more relevant than absolute values.

Shield Driver and Noise Considerations — To further mitigate external interference, the FDC1004 integrates a dedicated shield driver. The SHLD output mirrors the waveform and voltage potential of the CIN pins, ensuring no current flows between the sensor and shield. As a result, any capacitance between these nodes does not affect the charge-transfer measurement, significantly improving noise immunity.

I2C Communication — The FDC1004 is equipped with I²C for interfacing with an **MCU** serving as the master. Both devices need to operate at 3.3 V logic levels, which avoids the need to level shifting. From a hardware design perspective, this implies the use of standard I²C pull-up resistors on the SCL and SDA lines.

3.2.3.2 Capacitive Sensor Array Design & Sensing Surface

Capacitive sensing as a technology presents both an opportunity and certain challenges for the design choices. The sensing surface responsible for interaction with the user's input is on one hand capable of being obscured behind solid materials, assuming they have low dielectric constants (i.e plastic, plaster, dry wood, etc.), but on the other hand has very specific constraints for the design of the electrodes.

First of all, the number, size and positioning of the electrodes themselves is a key factor that needs to be analysed with the objectives of the module in mind. The arrangement and number of electrodes defines the type of gestures that can be detected, an array of greater complexity increases, by consequence, the complexity of the gestures that can be differentiated. The size of the electrodes will be the determining

factor on the sensing range of the module, the bigger the area the bigger the range. Next up there is a in depth analysis of these characteristics and respective design choice.

Electrode Geometry and Layout

The first design decision concerns the **number and spatial layout** of the sensing electrodes, as this directly determines the achievable spatial resolution for gesture detection. In general, a larger number of electrodes provides finer spatial discrimination; however, this must be balanced against system complexity and the number of available input channels. The selected capacitive-sensing IC, the FDC1004, offers four measurement channels, which naturally limits the architecture to four independent sensing electrodes.

Given the target application, the defined gestures require a two-dimensional sensing capability rather than a continuous high-resolution positional field. Therefore, a layout consisting of four electrodes arranged in a **cross pattern**—one element placed at each extremity of the horizontal and vertical axes—provides the necessary directional discrimination while fully utilising the available four channels. This configuration enables the system to detect changes in the electric field associated with hand motion along both axes, supporting the gesture vocabulary defined for the interface.

To guide the selection of electrode size, the documentation provided by the manufacturer includes empirical data relating **sensing area** to **proximity detection range**. Figure 2 and Table 2, reproduced from [34], demonstrate the expected sensing range for several square electrode areas when a human hand is used as the target. The detection threshold is defined as a capacitance variation of 3 fF relative to the baseline measurement. These results, obtained using a two-layer copper PCB electrode and shield, offer a practical reference for estimating the relationship between electrode area and achievable sensing depth.

Sensor area size (cm ²)	Proximity sensing range (cm)
0.25	15
1	17
4	19
9	22
16	25
25	28
36	31
49	33
64	37
81	39
169	50

Table 2: Proximity sensing range based on sensor area size, using data from [34].

Although the selected gesture set does not include longitudinal proximity-based actions (such as push or pull gestures), the system must still support **free-space gesture recognition**. Therefore, electrode size

cannot be arbitrarily reduced, as excessively small electrodes would diminish sensitivity to lateral motion and degrade directional performance.

Beyond sensing range considerations, several additional factors influence the determination of the **electrode geometry**. These include the need to optionally incorporate shielding structures, the spacing required between adjacent electrodes in the cross layout, and the physical constraint that the entire arrangement must fit within a 10 cm by 10 cm module.

In capacitive proximity sensing, electrode geometry—namely its area, perimeter shape, and spacing relative to neighbouring structures—plays a central role in shaping the electric field and determining measurement stability. Application notes from Texas Instruments emphasise that electrodes in the 15 mm to 25 mm scale provide an effective trade-off between sensitivity to hand-sized targets and manageable parasitic capacitances [35]. This scale also aligns well with the spatial limitations of the module, enabling four electrodes to be arranged symmetrically without significant field overlap. Consequently, an approximate electrode area of 300 mm^2 to 350 mm^2 is selected as an appropriate design target.

Consideration of Shielding

To improve directional resolution and suppress unwanted lateral coupling, the design further considers the inclusion of a **shield ring** surrounding each electrode. TI documentation recommends shield structures whose dimensions are comparable to, or moderately larger than, the sensing electrode, as this configuration enhances forward-focused field confinement without causing excessive attenuation [35]. Following this guidance, the shield ring diameter is increased by approximately 40% relative to the electrode diameter, balancing field shaping requirements with the available board area.

Finally, the separation between the sensing electrode and its shield ring must remain small but controlled. A spacing of 1 mm to 2 mm is widely cited as an effective range for preventing significant parasitic coupling while maintaining predictable fringing behaviour. This spacing is also practical from a fabrication perspective, ensuring that tolerances remain achievable while preserving the electrical characteristics required for free-space gesture detection. Below in Figure 8 is presented the final design of the electrodes and sensing surface.

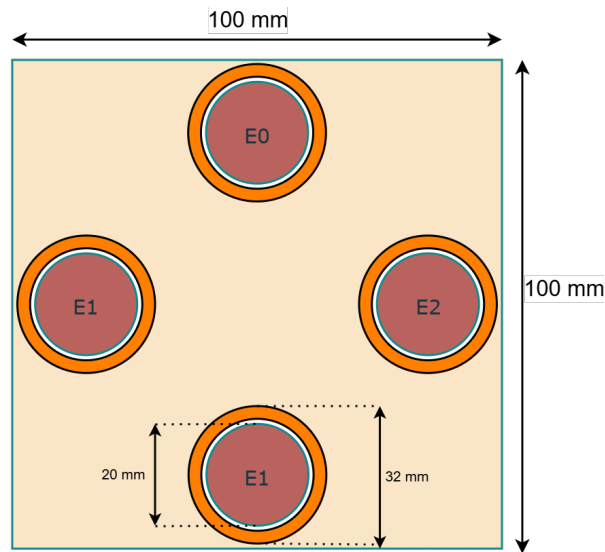


Figure 8: Electrodes and Sensing Surface Design

Together, these analyses define a coherent electrode geometry that satisfies the spatial constraints of the module, supports the required gesture vocabulary, and aligns with the design recommendations found in established capacitive sensing literature.

3.2.3.3 Processing Unit

First of all, as in previous sections, it is essential to revisit and clarify the specific requirements that this hardware component must address:

- Acquiring raw or preprocessed data from the capacitive sensing IC.
- Executing preprocessing steps (filtering, normalization, segmentation).
- Running the embedded machine learning model to classify gestures.
- Communicating recognized gestures to the higher-level system (e.g., Raspberry Pi or other controller).
- Supporting real-time operation.

With these aspects established, the selection of suitable hardware becomes the next design step.

Hardware selection and characteristics

From a design perspective, the processing unit must provide:

1. A compatible interface with the capacitive sensing IC, in this case I²C.

2. Integrated wireless connectivity to communicate with the Raspberry Pi-based control system.
3. Adequate computational resources to support local preprocessing and embedded machine learning inference.

Several microcontroller families could satisfy these requirements (e.g., STM32, nRF52, ESP32-based modules); however, the **ESP32-WROOM-32** development board was selected primarily because it was already available, thereby reducing cost and development time. Importantly, this choice still satisfies the identified hardware requirements for the intended prototype.

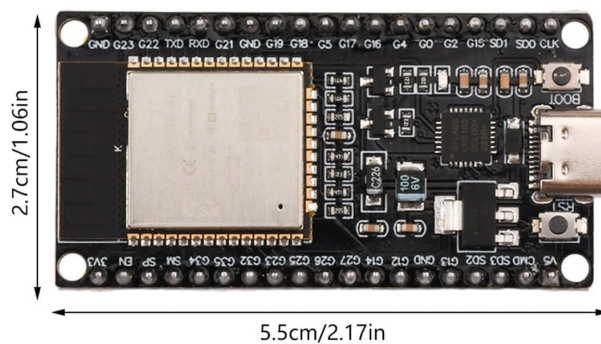


Figure 9: ESP32 Illustration

Key device characteristics

The specific device used (shown above in Figure 9) is a USB-C ESP32 development board built around the ESP32-WROOM-32 module and provides a convenient hardware platform for prototyping Wi-Fi and Bluetooth embedded applications:

- Chip: ESP32-D0WDQ6, Xtensa dual-core 32-bit LX6 microprocessor, up to 240 MHz
- Wireless: 2.4 GHz Wi-Fi and Bluetooth v4.2
- Memory: 448 kB ROM, 520 kB SRAM (plus RTC SRAM), and 4 MB SPI flash
- Electrical operating conditions: 3.0 V to 3.6 V supply voltage
- USB interface (board level): USB-C connector used for power and PC connectivity via a CP2102 USB-to-UART bridge (i.e., USB is used as a serial and programming interface)
- Controls and I/O breakout (board level): typical development-board implementation exposes most module pins on headers and provides BOOT (download) and EN (reset) controls for flashing over [Universal Asynchronous Receiver/Transmitter \(UART\)](#)

From this set of characteristics, compatibility with the defined system requirements can be confirmed. Furthermore, as shown in Figure 9, the physical size of the module is compatible with the predefined dimensions of the prototype, as the module occupies approximately half the length of the overall structure.

The choice of a development board instead of a bare ESP32 [System on Chip \(SoC\)](#) was made to simplify prototype development. In a future iteration of this project, the design of a dedicated [Printed Circuit Board \(PCB\)](#) would represent an appropriate and necessary improvement.

Electrical Integration

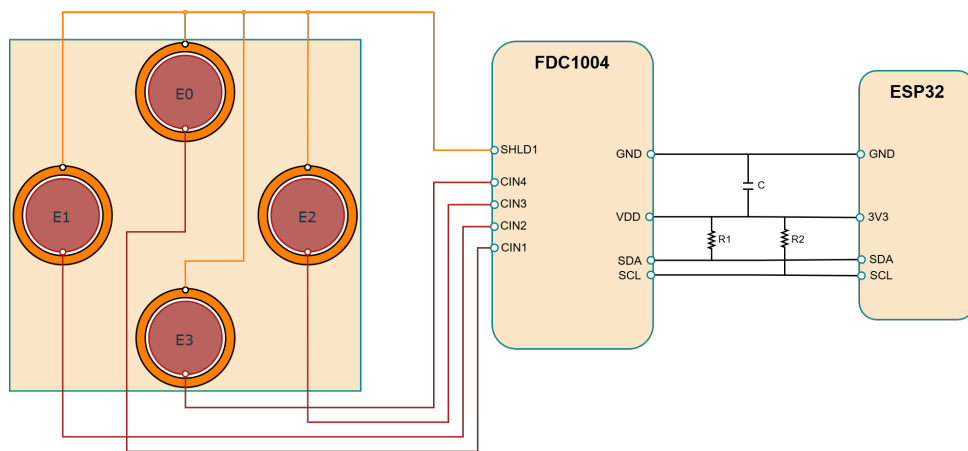


Figure 10: Electrical Connection Diagram

As seen in the diagram in Figure 10, the four electrodes presented in the sensing surface are connected to each channel pins present in the fdc1004. The ring shields around each electrode are all connected to the SHLD pin.

Between the fdc1004 and the ESP32 there is a simple circuitry for the I²C connections and the powering of the fdc1004 sensor. For the SCL and SDA connections there are pull-up resistors between each line and the 3.3 V/VDD pin, which are crucial for the functioning of the I²C protocol. A decoupling capacitor is placed between the GND and VDD lines.

3.2.3.4 Hardware Design Overview/Summary

3.2.4 Software Architecture

Similarly to the previous section, the focus here shifts to the software architecture. A detailed description of all software components present in the [GRM](#) is provided and serves as a guide for the subsequent implementation. All relevant software components target the ESP32, as it serves as the processing unit for the system.

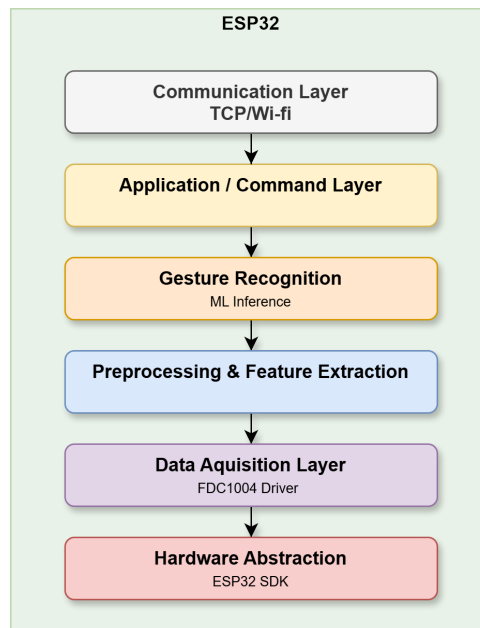


Figure 11: Software Architecture Layers Diagram

3.2.4.1 Software Architecture Overview

As presented abstractly in Figure 11, the main software layers targeted in this system can be identified. A more detailed description of these distinct layers is provided below:

1. A data acquisition layer that interfaces with the capacitive sensing IC.
2. A signal processing and preprocessing layer that prepares data for [ML](#).
3. A gesture recognition layer running the [ML](#) model.
4. A command abstraction layer that converts gestures into high-level events.
5. A communication layer that delivers these events to the external system.
6. A hardware abstraction layer that interacts directly with the ESP32 [Software Development Kit \(SDK\)](#).

3.2.4.2 Main Loop & State Machine Algorithm

Before addressing each software layer in detail, the coordinating and scheduling logic must be defined. The main loop invokes a [Finite State Machine \(FSM\)](#) to coordinate tasks such as data acquisition and [ML](#) inference. Figure 12 presents the flowchart of the system's main loop, followed by the state diagram; together, they describe the algorithm governing the main process.

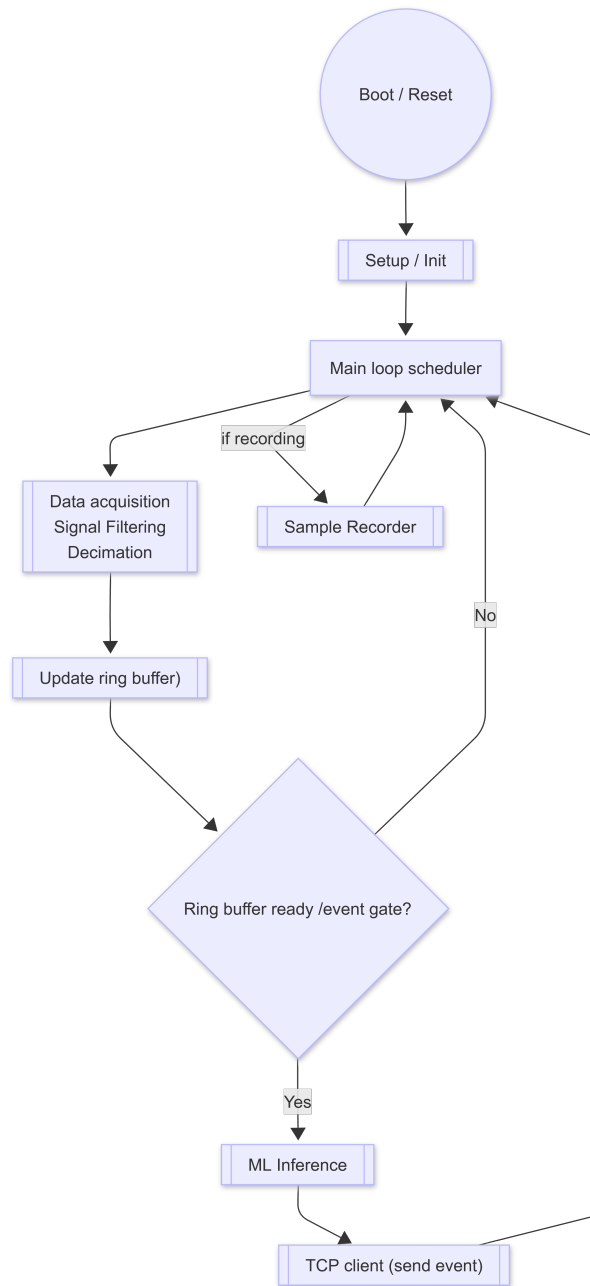


Figure 12: Main Loop Flowchart

In Figure 12, the main system logic is illustrated as a simple real-time loop that begins by acquiring raw sensor readings, transforming them into gesture events, and subsequently forwarding these events to the target system controller. After boot, the device initializes hardware (sensor and I²C), configures filtering, and establishes network connectivity. The main loop then samples the four channels, applies filtering, and continuously updates a 2 s ring buffer.

A lightweight event gate monitors this data stream for motion onset and offset. Only when a complete gesture is detected is the most recent window packaged and passed to the ML classifier, which is treated at this stage as a black box. The resulting labels are sent to the target system via a non-blocking TCP client,

ensuring that communication never delays sampling. Optional data recording and simple on/off controls integrate into the same loop without altering this execution flow.

3.2.4.3 Initialization & Setup Process

The setup routine executes once at boot or reset to initialize all hardware subsystems required for normal operation. Figure 13 illustrates the preparation steps performed during the setup phase.

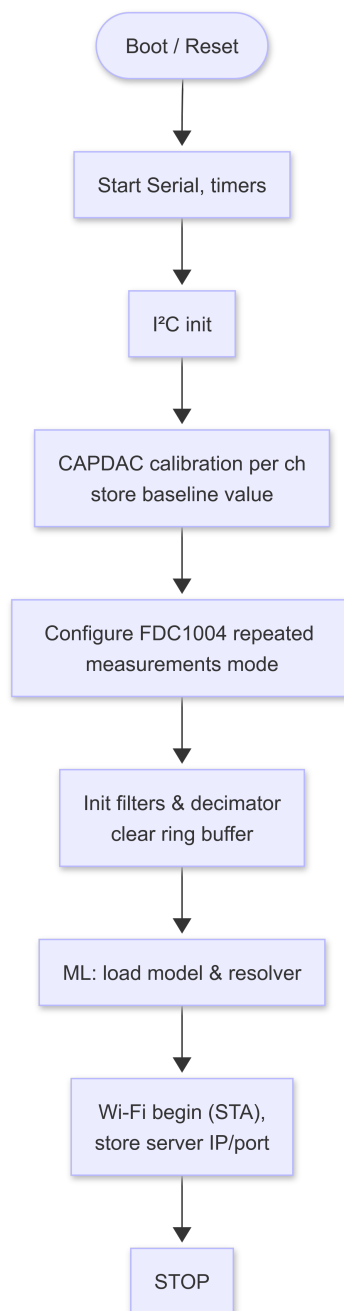


Figure 13: Setup process flowchart

1. **System I/O and timing:** The serial console is initialized for diagnostics and metadata output. Timing primitives are enabled to timestamp sensor reads and measure latency;
2. **I²C bus:** The ESP32 configures the I²C controller;
3. **Sensor calibration:** For each channel, CAPDAC tuning is performed by assigning an appropriate offset value. Baseline capacitance is measured and stored, and subsequent samples are baseline-subtracted to yield signed deltas;
4. **FDC1004 configuration for runtime operation:** The device is configured so that normal sampling requires only periodic reads:
 - Mapping each electrode to a measurement slot in single-ended mode to obtain four independent capacitance values;
 - Applying the calibrated CAPDAC value to the corresponding channel configuration register;
 - Selecting a conversion rate matching the firmware schedule (e.g., 100 Hz).
5. **Signal pipeline configuration:** The processing chain and decimation stage are initialized, internal buffers are cleared, and digital filter coefficients are reset;
6. **Inference runtime initialization:** The on-device model is loaded and the runtime prepared to classify fixed-size data windows.

To justify the selected sensor configuration, the FDC1004 is operated in repeated-conversion mode so that measurements are streamed at a fixed rate without per-sample reprogramming. In contrast, single-measurement mode would require configuration writes and trigger commands for each sample, increasing I²C traffic, CPU overhead, timing jitter, and the risk of missed deadlines at 100 Hz across four channels. Repeated-conversion mode enables stable timing, lower latency variance, and reliable synchronization with downstream processing.

3.2.4.4 Data Acquisition & Preprocessing Layer

The acquisition pipeline operates continuously within the main loop and is responsible for scheduled data acquisition, signal conditioning, and the production of normalized gesture windows suitable for inference (see Figure 12).

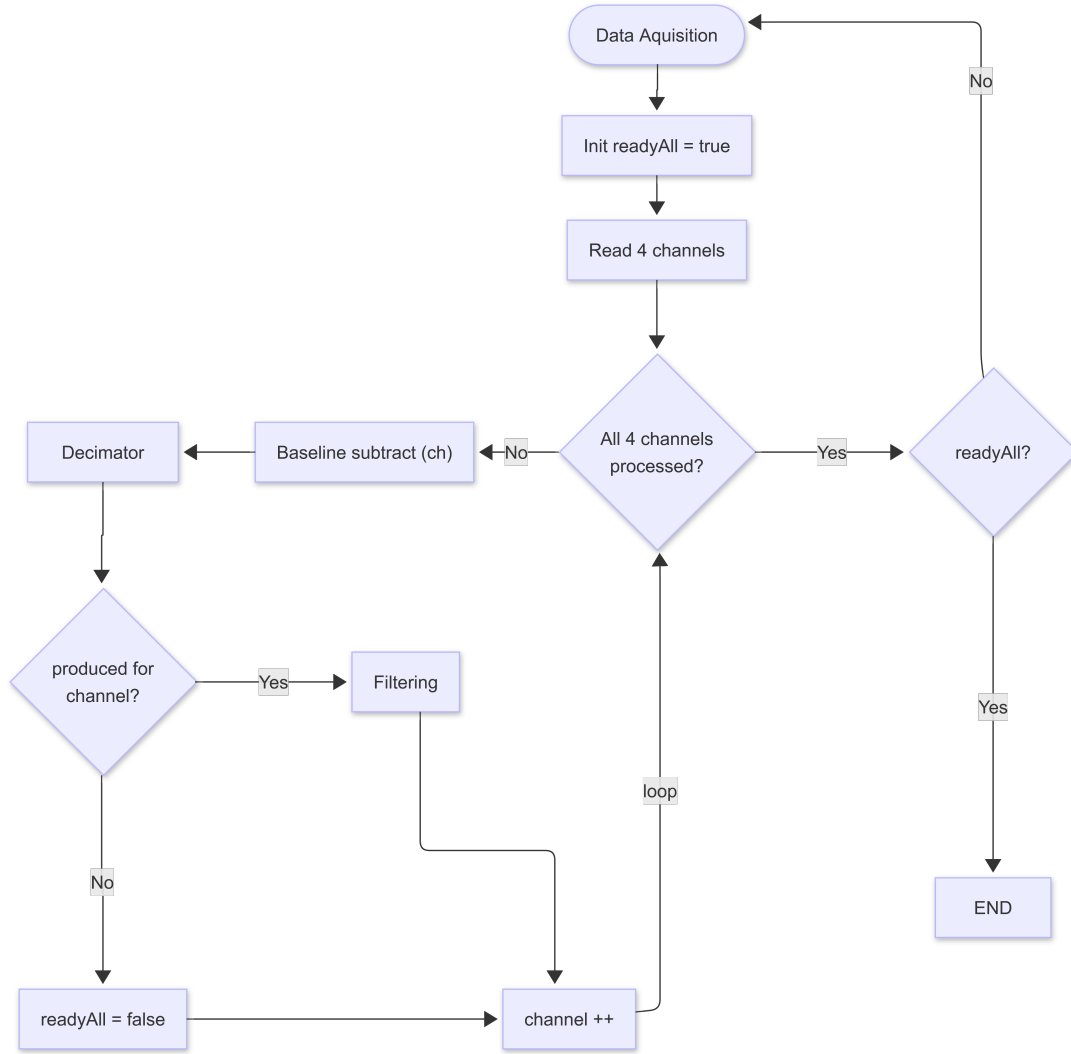


Figure 14: Data Acquisition Algorithm

As shown in the algorithm, the acquisition loop performs scheduled sampling at a fixed rate (e.g., 100 Hz). At each sampling instant, the firmware reads the four FDC1004 channels. Following acquisition, baseline subtraction is applied, converting each raw measurement into a delta value relative to its calibrated baseline, thereby centering the signals for subsequent processing.

For signal conditioning and rate reduction, a configured filtering pipeline suppresses unwanted components and reduces the effective sampling rate to a level appropriate for window-based inference. Specific filter parameters and decimation factors are detailed in the implementation section.

3.2.4.5 Windowing (Ring Buffer)

A sliding ring buffer of capacity N stores multi-channel samples, with each entry containing $C = 4$ channel values. The write index marks the next slot to be written; once the buffer is full, new samples overwrite the oldest entries in a circular fashion. For each decision, the buffer is read chronologically to form a window

of shape $[N \times C]$. The value of N is determined by the selected sampling rate and window duration and is specified in the implementation section.

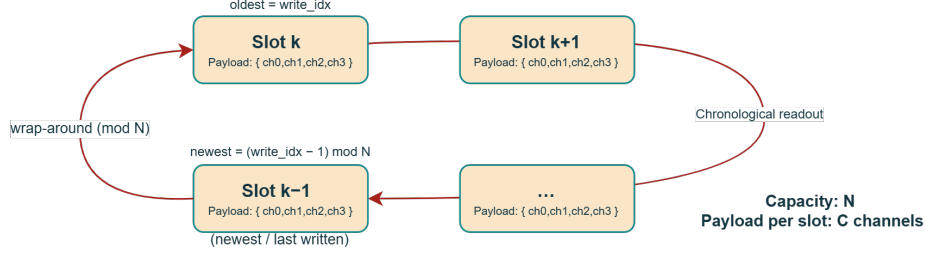


Figure 15: Sliding Ring Buffer Diagram

3.2.4.6 Event Detection (FSM)

Rather than classifying every frame, a lightweight event detector monitors a motion-energy score computed from recent inter-sample differences across channels and stored in the ring buffer. Using hysteresis and short hold counts (M, N), the detector transitions from IDLE to ACTIVE when energy exceeds a start threshold, and from ACTIVE to EMIT when it falls below an end threshold. After emission, a brief COOLDOWN state prevents duplicate triggers. At the EMIT boundary, the most recent 2 s window is normalized, quantized, and passed once to the ML model.

This scheduling aligns runtime behavior with dataset labeling, reduces false positives during idle periods, and bounds CPU and network usage.

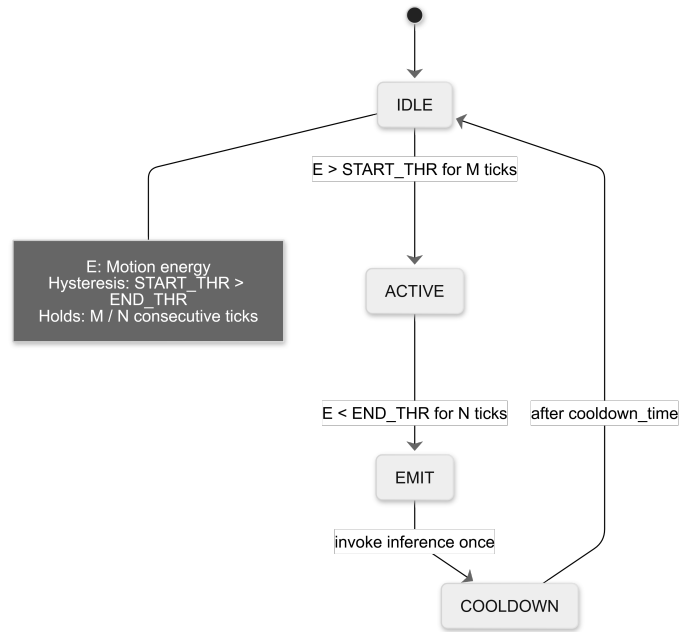


Figure 16: State Diagram

3.2.4.7 Gesture Recognition (ML Inference) Layer

At this stage, the ML component is treated as a black-box classifier producing a single discrete label per detected gesture. Its role is limited to mapping a fixed window of recent sensor data to a categorical decision. The inference routine receives a chronologically ordered buffer of shape $(N \times C)$, where $C = 4$. Prior to invocation, the buffer is standardized using per-channel statistics computed offline and linearly quantized to the model's `int8` input format. Model weights and intermediate tensors are statically allocated at startup, with no dynamic memory usage during runtime.

The inference routine returns a class index {up, down, left, right, none} and an associated confidence score derived from the quantized softmax output. If the predicted class is not none and the confidence exceeds a predefined threshold, a gesture event is emitted. In the event of setup-time faults, such as tensor-arena allocation failures, the system logs the error and halts safely.

3.2.4.8 Communication Layer

At the end of each loop iteration, a service routine maintains Wi-Fi and TCP connectivity and handles reconnections. Network operations are non-blocking and do not interfere with real-time scheduling. This structure ensures deterministic acquisition, stable inference windows, and clean event-based communication with downstream controllers.

Wi-Fi/TCP sockets were selected for communication with external controllers based on the requirements of the target actuator system, as discussed in a dedicated section.

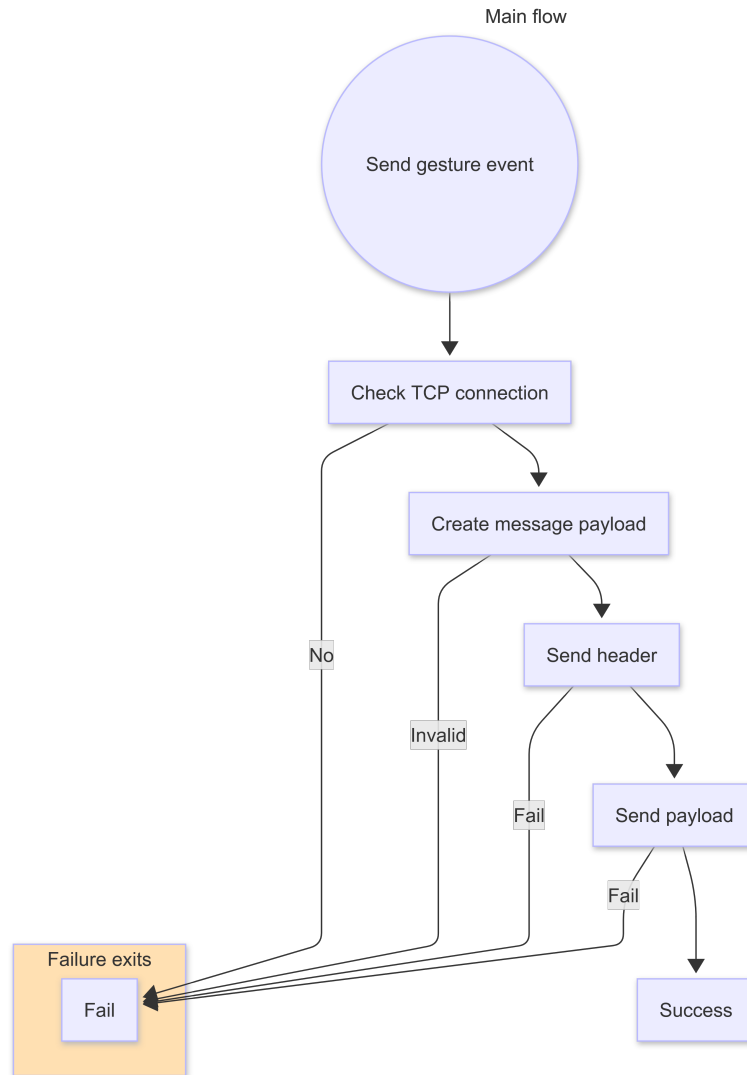


Figure 17: TCP Send Gesture Event

Figure 17 illustrates the TCP communication flow. After a gesture is detected, the main process invokes the communication routine, which verifies connection status, constructs the message payload and header, and transmits the event to the TCP server.

3.2.5 Embedded Machine Learning Integration

In order to achieve robust gesture recognition under varying users and environmental conditions, the gesture module integrates a machine learning (ML) classifier directly embedded on the ESP32. Rather than relying on fixed thresholds or purely rule-based logic, the system uses a data-driven model to map pre-processed capacitive signals to discrete gesture classes in real time. This subsection describes how the ML component is designed, trained, and integrated into the firmware running on the ESP32.

To keep the design stable while allowing empirical tuning during prototyping, we define interfaces and invariants here; concrete numeric values (sampling rate, decimation, thresholds) are finalized in the implementation chapter.

3.2.5.1 Design Assumptions and Invariants

Input Contract

Let $C = 4$ be the number of channels. The model input is a chronological tensor of shape $[K, C]$ (or $[K, C, 1]$ after reshape), produced at a decimated rate F_d obtained from the raw sensor rate F_{raw} by a decimation factor D . The downsampled feature frequency is defined in Equation 4, and the resulting input tensor is given in Equation 5.

$$F_d \triangleq \frac{F_{\text{raw}}}{D}, \quad (4)$$

$$X = [x_{t,c}]_{t=1..K, c=1..C} \quad (5)$$

Normalization Contract

Each channel is normalized according to Equation 6.

$$\tilde{x}_{t,c} \triangleq \frac{x_{t,c} - \mu_c}{\sigma_c} \quad (6)$$

where (μ_c, σ_c) are computed offline on the training corpus and embedded in firmware to guarantee training–inference parity.

The main design parameters used in the system design are summarized in Table 3.

Table 3: Design invariants and tunable parameters. Concrete values are finalized in the Implementation chapter.

Element	Symbol / Definition	Status
Raw sampling rate	F_{raw}	Tunable (implementation)
Decimation factor	D	Tunable (implementation)
Decimated rate	$F_d = F_{\text{raw}}/D$	Derived
Window length	K	Tunable (accuracy/latency trade-off)
Channels	$C = 4$	Fixed (hardware)
Normalization stats	(μ_c, σ_c)	Fixed after training
Gate parameters	$T_{\text{start}}, T_{\text{end}}, M, N, T_{\text{cool}}$	Tunable (validation)
Quantization target	INT8	Design choice

Role of ML in the Gesture Module

Still following the logic on the previous design section, the ML model sits in the pipeline, right after the pre-processing and before the command transmission. It is responsible for the classification of gestures from similar trajectories in the acquired data, to achieve this the model needs to be trained with an appropriate dataset that will permit the comparison and recognition in real-time. The is responsible for mapping pre-processed capacitive windows to discrete classes {up, down, left, right, none}.

Furthermore, its integration brings specific improvements, such as robustness and reliability against noise and small variations, to these types of application improving performance when compared to traditional simple thresholding and manual tuned rules.

Quantization strategy—design rationale (why INT8)

Full INT8 quantization is targeted for deployment on the ESP32 for four reasons:

1. **Memory and flash efficiency:** INT8 weights/activations reduce model and arena footprint by $\approx \times 4$ vs. float32, fitting MCU constraints without paging;
2. **Latency and energy:** integer convolution kernels are typically faster than software-emulated FP on Xtensa-class cores, lowering end-to-end latency and power;
3. **Determinism:** fixed-point arithmetic yields stable timing and avoids FP variability;
4. **Accuracy retention:** with a representative calibration set drawn from normalized training windows, INT8 commonly preserves classification accuracy for small CNNs.

3.2.5.2 Dataset Design (Concept and Interface)

This section specifies the dataset as a *design artefact*: what it must contain, how samples interface with the model, and the controls that ensure training–inference parity. Concrete numeric parameters (sampling rate, decimation, thresholds, counts) are finalized in the next section.

Objectives and Scope

- **Coverage:** classes $\mathcal{Y} = \{\text{up, down, left, right, none}\}$ across users, speeds, and distances.
- **Parity:** the same preprocessing/decimation/windowing and per-channel normalization used at inference are applied when creating training samples.
- **Separation:** subject/session-level splits to avoid leakage; a held-out test set remains immutable.

Acquisition and Labeling (Design-Level)

The samples are acquired using the same method previously designed, only alterations are made for the functioning to record samples instead of invoking the inference and transmitting. The recording sessions aim to label and record gesture samples of all gesture types and none natural segments, each session records one or more of gestures from the same type.

User Variability and Motion Diversity (Rationale)

Robust generalization requires the dataset to capture *who* performs a gesture and *how* it is performed. Capacitive trajectories depend on hand size, approach angle, stance, and habitual motion patterns; they also vary with speed profiles (quick vs. smooth), amplitudes, and partial or compound movements. If the dataset under-represents these factors, the classifier will overfit to a narrow execution style and degrade in real use.

Design goal — Encode variability explicitly so the learned decision boundary models the *gesture concept* rather than a single performer or canonical trace.

Dimensions of variability to cover —

- **User factors:** multiple individuals differing in hand size, dominant hand, height/reach, and typical motion cadence.
- **Execution styles:** slow/medium/fast swipes; short vs. long range; shallow vs. steep approach angles; deliberate vs. casual motions.
- **Boundary cases:** weak/partial gestures near the detection threshold; quick reversals; pauses mid-swipe; off-axis motions.
- **Context drift:** repeated recordings over time (temperature/humidity changes), minor sensor repositioning, and everyday environmental variations.

Expected benefit — A model trained on user- and motion-diverse data develops invariances to speed, amplitude, and idiosyncratic style, which (i) improves real-world accuracy, (ii) reduces false triggers on none, and (iii) stabilizes performance across deployments without per-user recalibration.

3.2.5.3 Deployment Framework Selection (TFLite Micro)

We adopt [TensorFlow Lite for Microcontrollers \(TFLM\)](#) as the embedded inference runtime for the ESP32, this choice follows from the constraints established before:

1. **Memory model and determinism.** TFLM uses a single statically allocated *tensor arena*, enabling fixed RAM footprint and predictable timing (no dynamic allocation during inference).
2. **Quantized execution.** Full INT8 quantization is natively supported (weights and activations), which reduces model size and latency while preserving accuracy for small models when calibrated with representative data.
3. **Tooling and ecosystem.** The TFLite converter and post-training quantization pipeline integrate with standard Keras/TensorFlow training, minimizing effort and improving reproducibility.

3.2.5.4 Data Representation and Feature Interface

Filter-agnostic preprocessing

The *Preprocessing* part is treated as a black box with a defined interface and goals. It converts raw samples at F_{raw} into a decimated stream at F_d that satisfies:

1. **Stationarity:** slow drift/baseline wander is removed so $\tilde{x}_{t,c}$ is well-centered.
2. **Bandwidth limiting:** out-of-band noise is suppressed to avoid aliasing before decimation by D .
3. **Scale stability:** dynamic range is consistent across sessions/users to keep normalization well-behaved.
4. **Latency budget:** the induced group delay is bounded and compatible with gesture-to-action latency.

Specific filter families and parameters (e.g., detrending, anti-alias low-pass, smoothing) are selected empirically during implementation to meet goals 1–4.

Windowing and normalization

A circular buffer maintains the most recent K decimated samples to form a chronological input window of shape $[K, 4]$. Normalization uses training (μ_c, σ_c) , applied *identically* at training and inference time to prevent covariate shift.

3.2.5.5 Model Selection and MCU-Oriented Optimization

Topology

The classifier is a compact 1D CNNs with two convolutional stages followed by dual global pooling and a small MLP head:

Table 4: Key tensor shapes (symbolic). Final K , f_1 , f_2 , h are fixed in Implementation.

Layer	Input shape	Output shape
Input	$[K, 4]$	$[K, 4]$
Conv1D(f_1)	$[K, 4]$	$[K, f_1]$
BN + ReLU	$[K, f_1]$	$[K, f_1]$
Conv1D(f_2)	$[K, f_1]$	$[K, f_2]$
BN + ReLU	$[K, f_2]$	$[K, f_2]$
GlobalAvgPool1D	$[K, f_2]$	$[f_2]$
GlobalMaxPool1D	$[K, f_2]$	$[f_2]$
Concat	$[f_2], [f_2]$	$[2f_2]$
Dense(h)	$[2f_2]$	$[h]$
Dense(5)+Softmax	$[h]$	$[5]$

In Table 4 K is the window length (design tunable), f_1 , f_2 are the numbers of filters in the two Conv1D layers, and h is the hidden size in the penultimate Dense layer.

Design Rationale

- **Early temporal feature extraction.** Two shallow Conv1D layers capture short-range temporal patterns per channel and progressively build more abstract features with modest compute cost.
- **Shift robustness.** *Global* average and max pooling over time remove phase sensitivity and make the classifier invariant to small temporal shifts in gesture execution. Using both mean and max captures complementary statistics (overall activation vs. salient peaks), improving separability at negligible cost.
- **Compact head.** A single hidden Dense layer strikes a balance between capacity and latency/flash footprint; the final Dense(5) matches the class set {up, down, left, right, none}.
- **Regularization.** Dropout is included during training to reduce overfitting; at inference it is a no-op, so it does not affect runtime.

Deployability on TFLite Micro (operator mapping).

The graph is intentionally chosen to compile cleanly on TFLM:

1. **Conv1D $\rightarrow$ Conv2D kernel.** TFLM implements 1D convolution via CONV_2D on a reshaped input $[1, K, 4, 1]$ with a kernel of size $(k_t, 1)$. The converter handles this mapping; memory planning remains static.

2. **BatchNorm folding.** At inference, Batch Normalization is folded into preceding Conv1D weights and biases (scale/shift absorbed), so no BN kernel is needed on device and no extra arena is consumed.
3. **Global pooling.** GlobalAveragePooling1D and GlobalMaxPooling1D lower to MEAN and REDUCE_MAX reductions across the time axis, both supported by TFLM. Because the reduction is global, no stride/windows need to be stored.
4. **Concatenate/Dense/Softmax.** Concatenate is supported; Dense maps to FULLY_CONNECTED; Softmax has a quantized kernel. Dropout is removed at export time for inference graphs.
5. **Shape ops.** Any layout changes are expressed with RESHAPE (constant-time), avoiding converter-only ops such as EXPAND_DIMS that lack micro kernels.

Resource targets and invocation policy

A single MicroInterpreter instance with a fixed tensor arena runs on the ESP32. The interpreter is invoked *once per gesture* at gate emission (see Figure 12), which bounds CPU usage and avoids repeated, correlated predictions during rebounds. Final arena size, model size, and measured latency are reported in the Implementation chapter.

3.3 Integration with Actuator Systems

Having defined the architectural design of the GRM and its main components, now there is the need to design the integration process with a target system for actuation purposes. The first step in this process is to define the target system and its architecture. This system was developed as a prototype development integrated into the project R2U Technologies | modular systems developed by the Algoritmi team which encompassed this dissertation's project.

3.3.1 Target Actuation System: LED Wall Architecture

The system is a prototype for a smart wall system with different types of purposes including led lighting and heating surface (Ambient and Radiant, respectively) integrated in its structure and with a control system developed in order for a user to control it via an android APP. Below, in Figure 18, is a diagram depicting its architecture.

From this whole structure it is relevant to establish that the *led panel* is the main target for this project's system, so the GRM module will interact with this system with that specific panel as actuator.

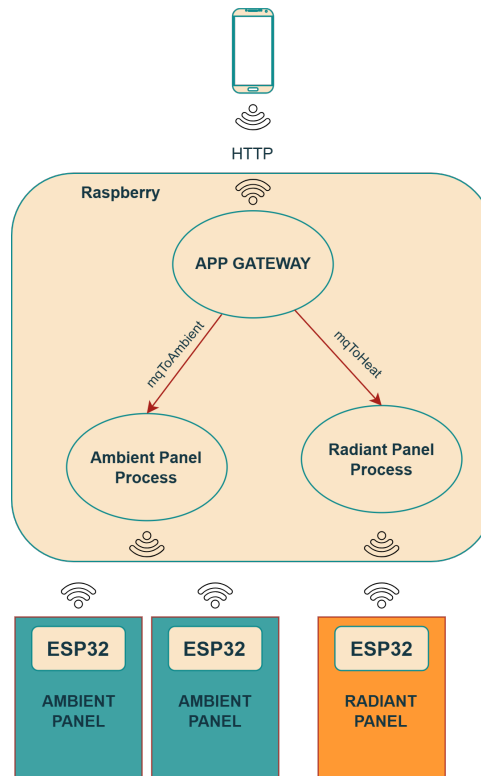


Figure 18: Target System Architecture

As seen in Figure 18 the system is equipped with a Raspberry Pi serving as a general system manager, it is responsible for running multiple processes which divide all its functions into different sections:

- **Gateway Process:** This process is responsible for establishing communication with APP via HTTP. It hosts a flask server that receives HTTP requests and separates the requests based on panel type and forwards them to the respective panel process.
- **Panel Processes:** These processes are directly responsible for coordinating the communication with all panels of its type, it establishes TCP communication with the panel controllers (ESP32) and coordinates/synchronizes the functionalities applied to the panels.

On the panel itself the MCU utilized is the ESP32 that is responsible for the low-level control of the panels functionalities, which (in the ambient panel's case) consists of the digital mapping and control of up to 10 WS2812 LED strips. Additionally it is able to receive commands from the raspberry's panel process utilizing TCP socket communication, it deciphers, applies the respective command and returns a response. The physical panel in itself is a 300cm by 60cm structure that is endowed with the hardware previously mentioned.

3.3.2 Interface Between Gesture Module and Actuator System

With the target system established, we detail how the Gesture Recognition Module (GRM) integrates with the existing control stack to drive the LED wall. The integration preserves the system's current pattern: per-panel processes on the RPi communicate with ESP32 controllers over TCP, while the RPi's gateway (Flask server) arbitrates commands via message queues.

The GRM reuses this pattern by adding a dedicated RPi process (*Gesture Module Process*) that accepts TCP messages from the GRM's ESP32, parses gesture events, maps them to LED-wall commands, and forwards those commands into a dedicated gesture queue.

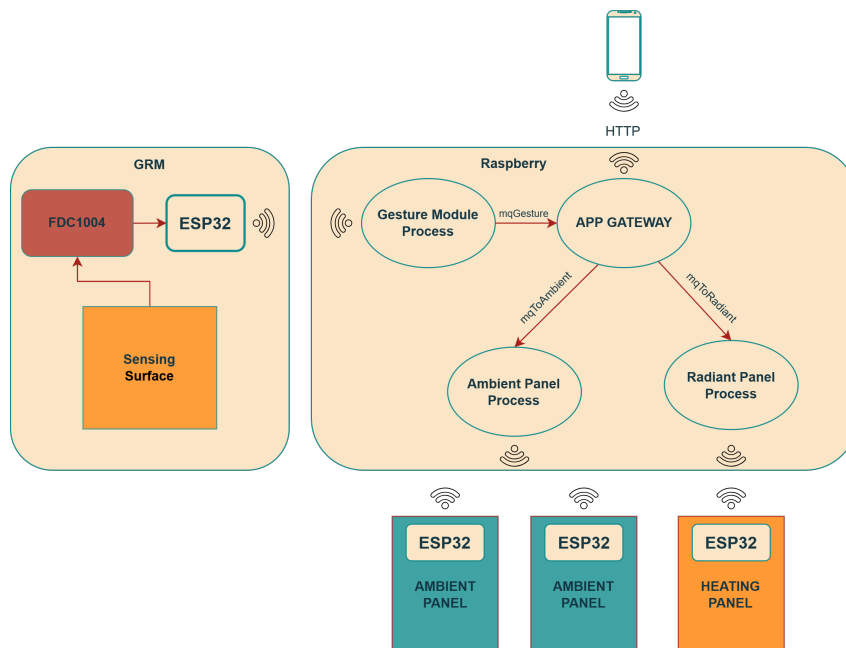


Figure 19: Integration of the GRM into the existing control stack. The new Gesture Module Process mirrors other panel processes and uses the same gateway queue.

Transport and Framing Protocol

Transport — ESP32 (GRM) acts as a TCP client connecting to the Gesture Module Process (TCP server) on the RPi. Connections are long-live with keep-alives enabled.

Framing — Application messages are framed as 2-byte big-endian length followed by a UTF-8 JSON payload:

```
[len_be:2][json:len]
```

This avoids ambiguity from partial reads and concatenated frames.

Direction and cadence — Traffic is primarily *upstream* (ESP32→RPi) and *event-driven*: the ESP32 sends one message per emitted gesture. No polling is required. Optional heartbeats are permitted but not required.

Validation rules (RPi) —

- Reject if type $\neq$ "gesture_evt".
- Drop silently if gesture="none" or $\text{conf} < \tau$ (deployment threshold).
- Accept only whitelisted gesture values; log unexpected values.
- Enforce simple rate-limits per device to avoid flooding: at most one accepted event per cooldown window.

Gesture Module Process

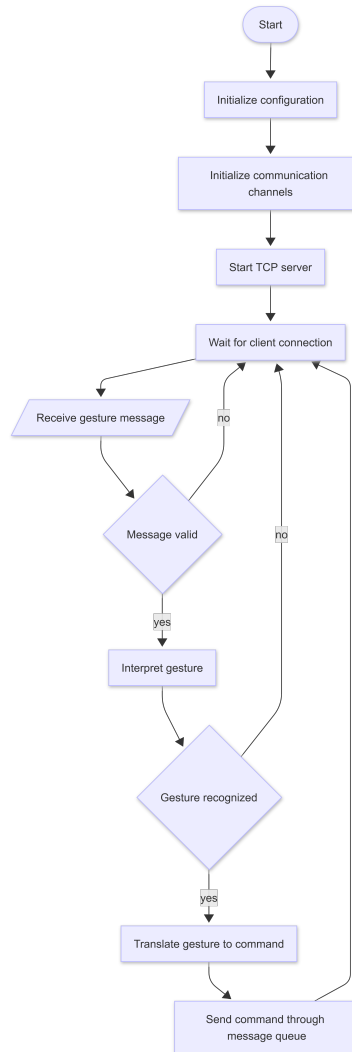


Figure 20: Algorithmic representation of the Gesture Module Process on the RPi.

The process illustrated in Figure 20 is a single-threaded event loop with the following responsibilities:

1. **Accept:** listen on the configured TCP port; accept exactly one ESP32 client at a time (reconnects allowed).
2. **Read:** receive framed JSON messages; each complete frame is parsed.
3. **Interpret:** validate fields, apply confidence threshold τ .
4. **Map:** convert gesture to a canonical LED-wall command (Table 5).
5. **Forward:** enqueue the command to the gateway's existing /mqGesture (non-blocking; short retry/backoff when full).

Gesture-Command Mapping

An important part of the process is the translation from gesture semantics to actuator commands. The default mapping targets user expectations and minimizes error impact (e.g., an accidental up becomes a small brightness step rather than a mode switch). Table 5 shows the baseline mapping.

Table 5: Gesture-to-command mapping for the LED wall

Gesture	Command / Action
Swipe Up	brightness_delta $\rightarrow$ +s
Swipe Down	brightness_delta $\rightarrow$ -s
Swipe Right	next_effect
Swipe Left	prev_effect

Operational Considerations

Startup and shutdown — On startup, the Gesture Module Process attempts to open the gateway queue (retry on EWOULDBLOCK until the gateway is up), then begins listening on TCP. On shutdown (SIGINT), it closes client and server sockets to interrupt any blocking accept/recv, then closes the queue handle (no unlink; the gateway owns the queue).

Resilience —

- **Network loss:** if the TCP client disconnects, the process returns to accept and logs a reconnect event.
- **Malformed payloads:** invalid JSON or missing fields are ignored with a single-line warning.
- **Queue saturation:** non-blocking send with short retry; coalesce deltas; drop with metric increment if persistence fails.

3.3.3 Example Gesture-to-Action Scenarios

Scenario A: Single upward swipe (brightness increase) —

1. The GRM emits: `{"type": "gesture_evt", "gesture": "up", "conf": 0.89, ...}`.
2. The RPi validates ($\text{conf} \geq \tau$), maps to `{"cmd": "brightness_delta", "value": "+s"}` and enqueues to `/gesture_queue`.
3. The gateway forwards to the LED-wall process; brightness increases by s units. Total path is event-driven; no polling delay is incurred.

Scenario B: Rapid double-swipe up (coalesced brightness) —

1. Two up events arrive within $\Delta t < 200$ ms.
2. Queue temporarily full; first enqueue succeeds, second sees EAGAIN.
3. The process coalesces the second into a net delta and retries once; if still full, it drops with a metric increment. The net user effect is either +2s or +s depending on queue health, but never a stall.

Scenario C: Low-confidence gesture —

1. The GRM emits `gesture="right"` with `conf = 0.31`.
2. The process rejects below τ and logs a filtered event; no scene change occurs.

Scenario D: Rebound suppression —

1. The GRM gate ensures only one event per physical gesture; rebounds do not produce extra `gesture_evt` messages, keeping the RPi side stateless and simple.

3.3.4 Generalization to Other Actuators (Beyond LED Walls)

The interface is intentionally actuator-agnostic. Two abstractions make reuse straightforward:

1. **Semantic mapping layer.** Replace Table 5 with a per-actuator mapping (e.g., “up” $\rightarrow$ `volume_delta:+1`, “right” $\rightarrow$ `next_track`) while keeping the incoming gesture schema unchanged.
2. **Target addressing.** The `panel_cmd` envelope includes a `target` field. New actuators register a unique `target` and a command vocabulary; the gateway routes by `target`.

Examples include: HVAC setpoint (`temp_delta`), media controls (`play/pause/next`), blinds (`position_delta`), each with their own rate limits and safety bounds.

3.4 Design Summary

This section summarizes the design section, consolidates the architectural choices, module contracts, and integration patterns defined for the Gesture Recognition Module (GRM) and its interface with the existing Raspberry Pi (RPi) actuator stack.

3.4.1 Architecture at a Glance

- **Processing path (ESP32):** acquisition $\rightarrow$ *Preprocessing block* (filter-agnostic, with decimation) $\rightarrow$ fixed window $[K, 4]$ $\rightarrow$ embedded ML inference (INT8, TFLite Micro) $\rightarrow$ event-driven transport.
- **Event contract:** exactly one label per physical gesture via a finite-state *gate* (IDLE $\rightarrow$ ACTIVE $\rightarrow$ EMIT $\rightarrow$ COOLDOWN).
- **RPi integration:** a dedicated *Gesture Module Process* accepts framed TCP messages from the GRM, maps gestures to LED-wall commands, and forwards them to the gateway's existing server queue.

3.5 Design Validation and Summary

This section closes the design chapter with a concise recap of *what was designed*, *why*, and *what remains to be fixed in implementation*.

3.5.1 What the Design Establishes

- A clear processing path on the ESP32: acquisition $\rightarrow$ preprocessing (black box with stated properties) $\rightarrow$ fixed window $[K, 4]$ $\rightarrow$ embedded INT8 inference $\rightarrow$ event-driven transport.
- A one-shot emission policy via a simple gate (IDLE $\rightarrow$ ACTIVE $\rightarrow$ EMIT $\rightarrow$ COOLDOWN) to ensure at most one label per physical gesture.
- An MCU-oriented model topology (Conv2D with early cross-channel mixing, global pooling, Dense) restricted to TFLM built-ins.
- A minimal, framed TCP interface (length-prefixed **JSON**) into a Raspberry Pi *Gesture Module Process* that maps gestures to the existing gateway command vocabulary.

3.5.2 Why These Choices

- **Preprocessing as a contract:** allows empirical tuning on hardware while guaranteeing stationarity, band-limit before decimation, scale stability, and bounded delay.
- **Fixed input contract $[K, 4]$ + z-score:** preserves training–inference parity.
- **Shallow Conv + pooling:** time-shift tolerance with small compute/memory footprint suitable for ESP32.

- **Event-driven I/O**: avoids continuous inference and network chatter; integrates cleanly with the gateway.

3.5.3 Interfaces Summarized

- **ESP32** → **RPi**: length-prefixed **JSON** `gesture_evt` (device ID, timestamp, gesture, confidence).
- **RPi** → **Gateway**: `panel_cmd` envelope (target, command, optional value), reusing the system's existing queue and routing.

3.5.4 Constraints and Assumptions

- Single ESP32 loop with non-blocking networking and one interpreter instance.
- Class set fixed to {up, down, left, right, none}.

3.5.5 Deferred to Implementation

The design keeps numeric choices open for evidence-based tuning on the prototype:

- Sampling/decimation parameters (F_{raw}, D, F_d) and concrete filters (family, cutoff, order, measured delay).
- Window length K , gate thresholds/holds, confidence threshold τ , and actuator step size s .
- Measured budgets: inference latency, arena RAM, CPU usage, and end-to-end latency to the gateway.

Chapter Summary

In short, the chapter defines stable *interfaces* (input window, event, transport) and *policies* (one-shot emission, non-blocking operation) while leaving tunable parameters to implementation. This separation enables reproducible integration with the existing **RPi** gateway and provides a clear path to finalize numbers based on measurements on the physical prototype.

Chapter 4

Implementation of the Gesture Recognition Module and Prototype System

This chapter describes the practical realization of the gesture recognition system presented in the previous chapter. While the previous chapter focused on the conceptual design, architectural definition, and justification of the chosen approaches, the present chapter details how these concepts were translated into a working prototype. The implementation of the hardware, firmware, and supporting software components is described, including the concrete configurations, tools, and technologies adopted.

The chapter begins with the physical and electronic implementation of the capacitive gesture recognition module, sensing electrodes, enclosure, and their integration. It then describes the embedded firmware and machine learning model deployed on the microcontroller, followed by the implementation of the surrounding prototype system, namely the Raspberry Pi server, communication mechanisms, and LED wall controller. Finally, the complete end-to-end setup is outlined, illustrating how all components operate together as a coherent prototype that demonstrates the capabilities and integration potential of the proposed gesture recognition module.

4.1 Gesture Module Hardware and Physical Structure

The module is a two-panel stack: a *back panel* that carries the ESP32 and the FDC1004 circuitry on a two-sided protoboard, and a *front panel* that implements the sensing surface with round electrodes and concentric shield rings. The two panels are linked by shielded cables that pass through a central hole.

4.1.1 Layered Structure (Back Panel + Front Panel)

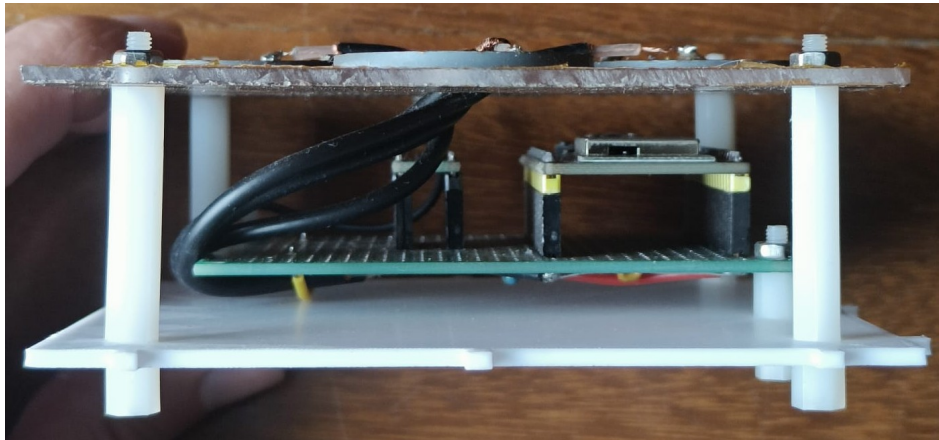


Figure 21: Layered structure of the module

From Figure 21, the layered structure is shown clearly: the front panel includes the round electrodes and ring shields, with shielded cables soldered onto it. It can also be seen that a layer of dielectric tape is used for insulation. The backside of this top layer is covered with the same copper tape used for the electrodes and shields, and is connected to the SHLD pin, as are the ring shields surrounding the electrodes.

4.1.2 Protoboard and Circuit (Back Panel)

On the back panel, the double-sided protoboard is screwed in place and contains the following soldered components (see Figure 22): (i) an ESP32 module, (ii) an FDC1004 on a small breakout board (soldered on the top side), (iii) and only the following connections on the bottom side: SDA and SCL with **two** 10 k Ω **pull-up resistors** to the ESP32 3.3 V pin, one **decoupling capacitor** for the FDC1004 supply, the 3.3 V and **GND** rails to the FDC1004, **shielded CH1–CH4 cables**, and the **shield lines** from the FDC1004 SHLD driver to the ring shields.

Power Supply — Power is provided via **USB 5 V** into the ESP32. The **FDC1004 is powered from the ESP32 3.3 V pin**. No external regulators or supplies are used.

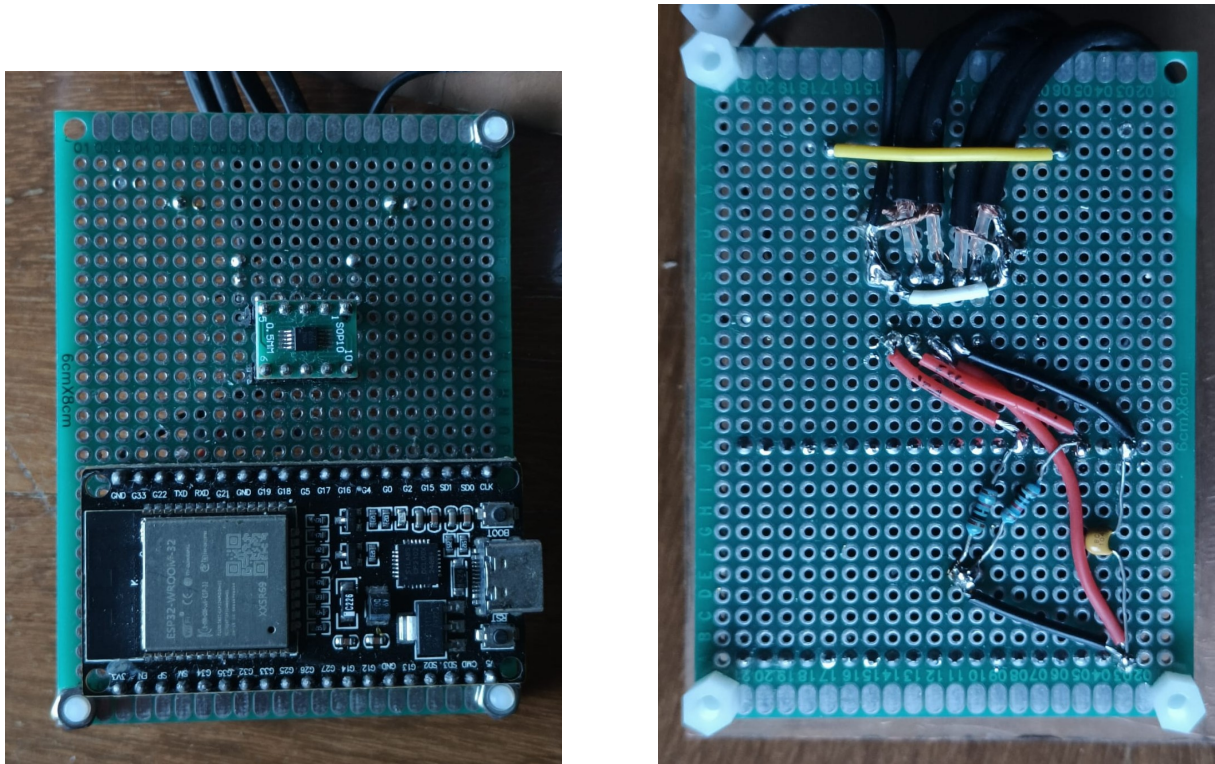


Figure 22: Left: top side with ESP32 and FDC1004 breakout, Right: Connections between ESP32 and FDC1004

Table 6 summarizes this connections.

Table 6: Back panel connections (as built).

Net	Connection	Note
USB 5 V	ESP32 VIN/5V	module power
3.3 V	ESP32 3.3 V → FDC1004 VDD	FDC1004 supply
GND	ESP32 GND ↔ FDC1004 GND	common ground
SDA	ESP32 GPIO21 ↔ FDC1004 SDA	10 kΩ pull-up to 3.3 V
SCL	ESP32 GPIO22 ↔ FDC1004 SCL	10 kΩ pull-up to 3.3 V
Decoupling	FDC1004 VDD–GND	single cap (placed near IC)
CH1–CH4	FDC1004 INx ↔ front panel electrodes	shielded cables
SHLD	FDC1004 SHLD → ring shields	shield driver line

4.1.3 Sensing Surface with 3D–Printed Electrodes and Shield Rings (Front Panel)

Layers on the front panel —

1. **Underside:** covered in copper tape, connected to the **FDC1004 SHLD driver**.
2. **Front side base:** a thin layer of dielectric tape.

3. **Electrodes + rings:** 3D-printed plastic forms (round electrode and concentric shield ring) used as stencils to apply copper tape to the intended shapes; glued to the front panel.
4. **Cable feedthrough:** shielded cables from the back panel pass through the **central round hole**; the inner conductor is soldered to the **electrode**; the cable's outer braid/shield is soldered to the **ring shield**.

These details are explained visible in Figure 23 below.

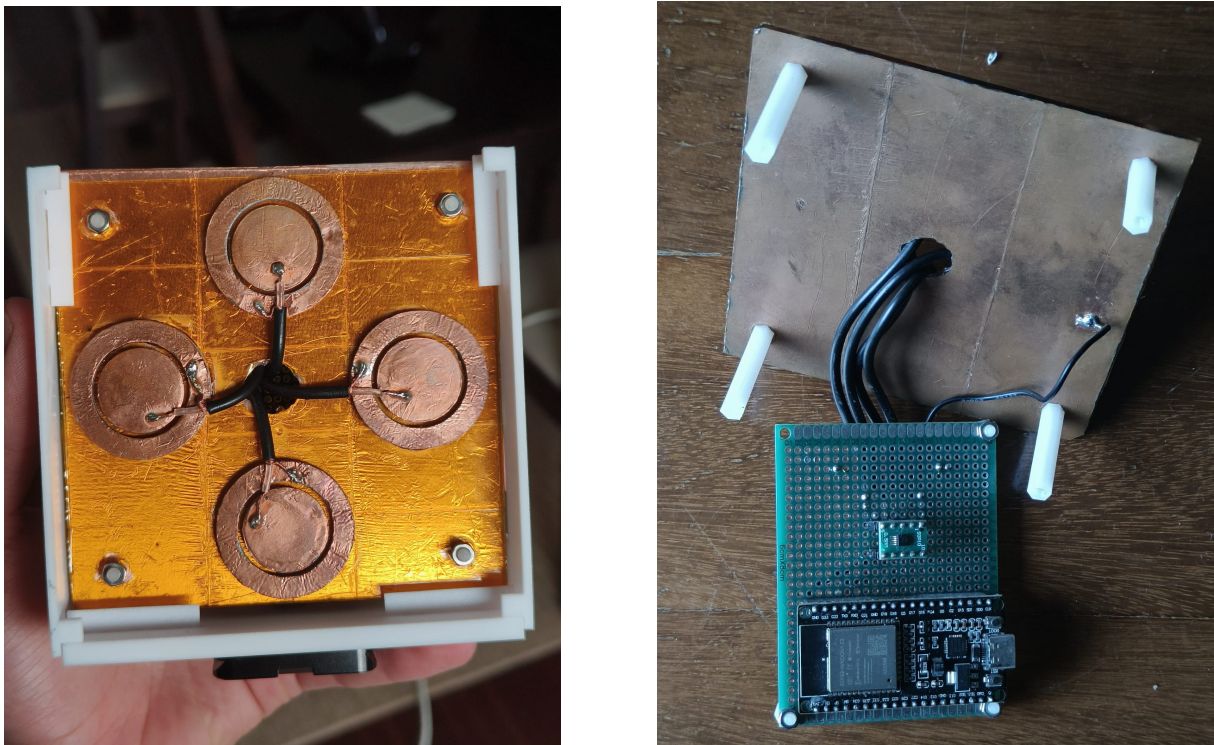


Figure 23: Front panel. Left: front side with round electrodes and shield rings. Right: underside of the sensing surface

The front side, with round electrodes and shield rings covered with copper tape, was produced using 3D-printed parts as stencils to ensure repeatable shapes. The underside is fully covered with copper tape connected to the SHLD driver to minimize noise and interference. On the front, a dielectric tape layer (for insulation) is visible beneath the electrodes, which were glued on top of it.

The electrodes and ring shields were printed out of 3D models that are visible in Figure 24

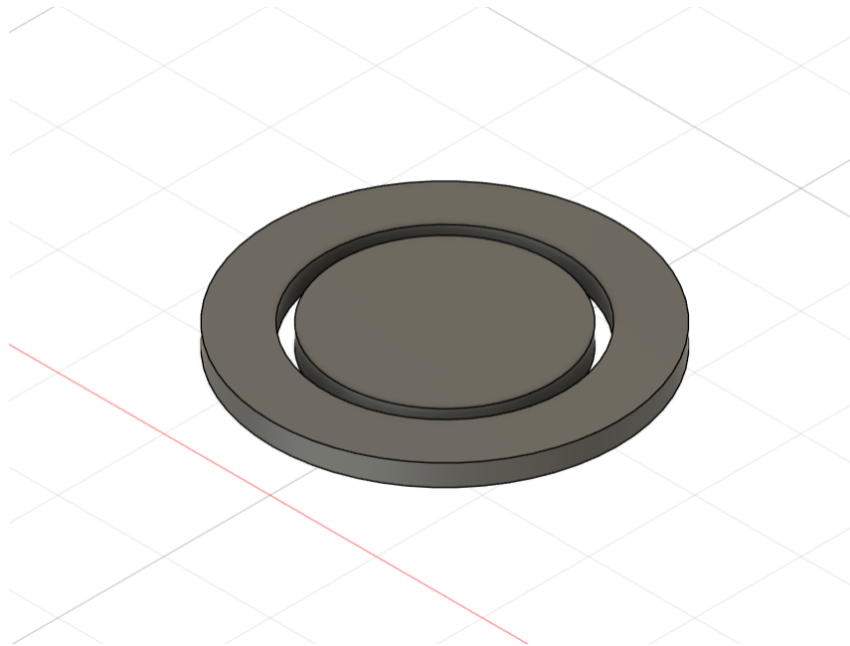


Figure 24: 3D CAD of electrode and ring-shield forms used to shape the copper tape and ensure repeatable geometry.

4.1.4 3D–Printed Outer Structure and Front Panel Mount

All structural parts are 3D–printed. Notable features: (i) one side has an **opening for the USB power cable** to the ESP32; (ii) small **side guards** guide a square “plaster” cover that **slides in** from the top to hide the module (front slot is open to allow sliding in/out); (iii) the **bottom plate is thicker** and has a **central hole** to accept a **female threaded insert** (tripod mount) to hold the module at any height.

4.1.4.1 CAD Overview and Photos

Figure 25 illustrates the entire module’s outer shell in digital 3D parts.

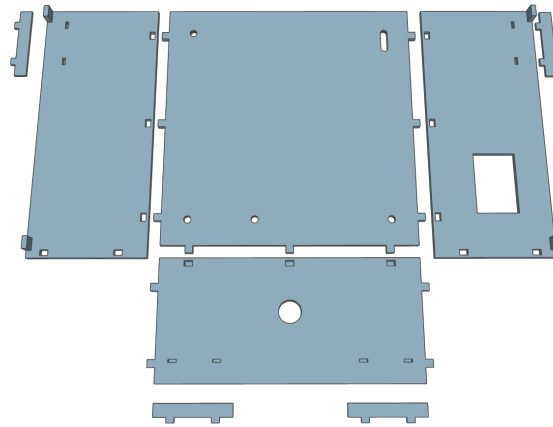


Figure 25: 3D CAD of the outer structure

4.1.4.2 Real Module Assembly

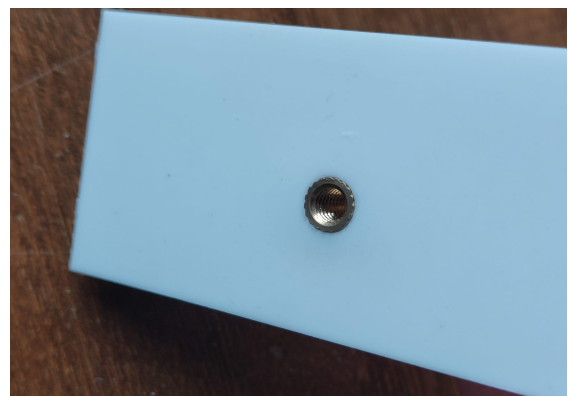


Figure 26: Real assembly. Left: module mounted with sliding cover in place. Right: Female screw for tripod

In Figure 26, the sliding front cover can be seen on the left side; this is enabled by the side guards and the open top. The right image shows the female screw insert used to attach the mounting piece presented below:



Figure 27: Tripod mount piece

With this, the structure is finalized and mounted on a tripod to enable adjustable height for testing, data recording, and operations (see Figure 27).

Front Panel — The front panel material was intended to be plaster to emulate a modern interior wall. Due to difficulty in acquiring this material in the required proportions, a square foam-board piece (K-Line) was used instead. With a selected thickness of 5 mm, even though gypsum/drywall typically used in homes has a higher dielectric constant, the difference is not substantial compared to the K-Line used (typically < 1.5).

4.2 Firmware and ML Implementation on the ESP32

This section details the software stack on the ESP32 and Raspberry Pi and the complete ML pipeline from dataset preparation to on-device deployment. All previously deferred parameters are fixed here with evidence (plots, logs, measurements).

4.2.1 Codebase Overview and Build Tooling

Repositories and layout

The firmware, training/export utilities, and the Raspberry Pi process are organized as shown in Fig. 28–30. This subsection only orients the reader; technical parameters are fixed in the next subsections before dataset creation.

- **src** — ESP32 firmware sources.
- **include** — public headers used by `src/`.
- **lib** — vendored third-party library (FDC1004).
- **testing** — Python tools for capture, dataset build, training, conversion, and quick visualization.

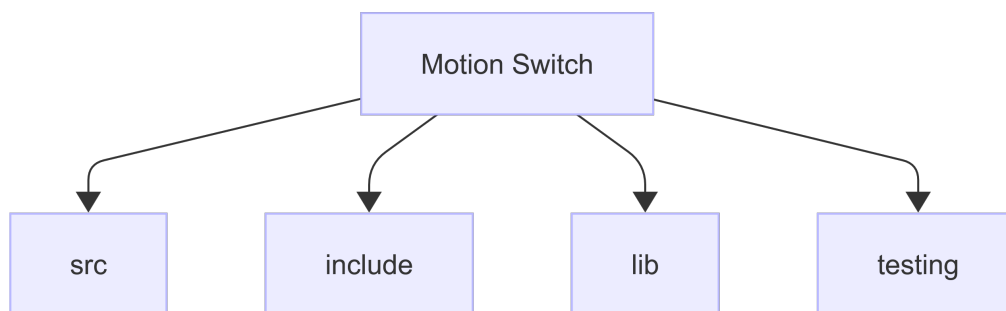


Figure 28: Top-level repository layout.

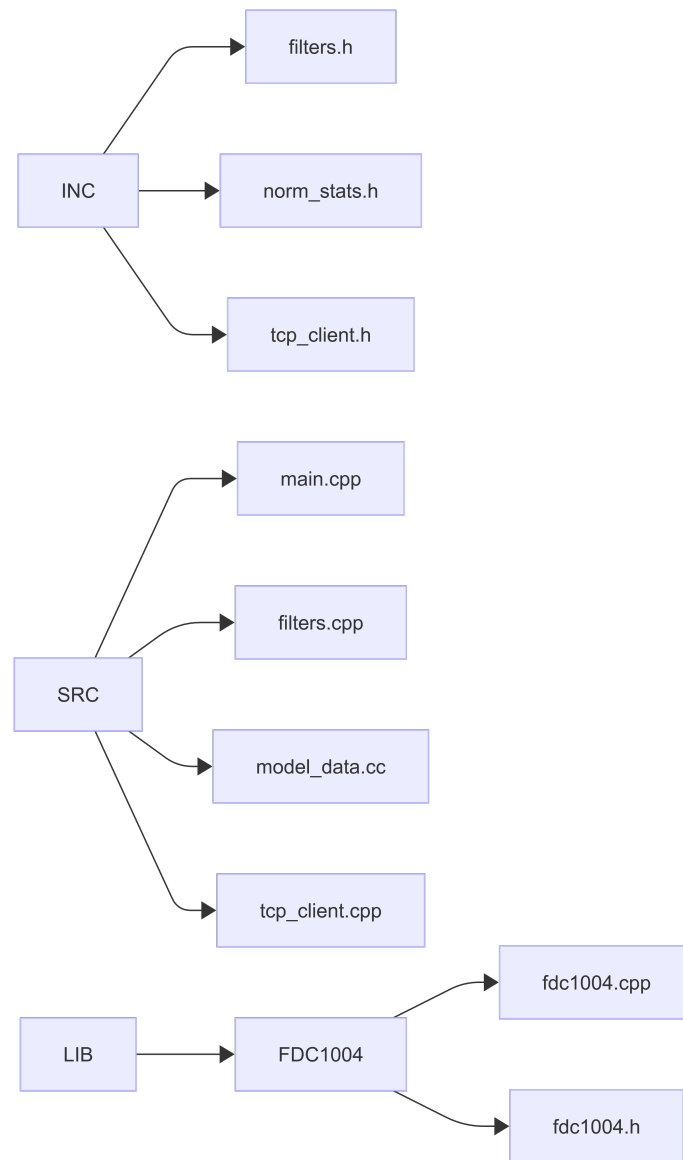


Figure 29: Firmware tree: headers in `include/`, sources in `src/`, FDC1004 driver in `lib/`.

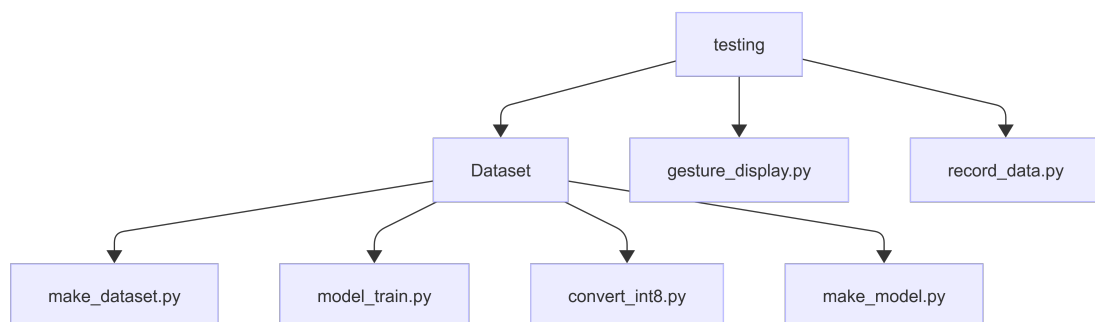


Figure 30: Testing utilities for capture, dataset build, training, conversion, and visualization.

File roles

- `src/main.cpp` — 100 Hz read $\rightarrow$ preprocess/decimate $\rightarrow$ ring $[K, 4]$ $\rightarrow$ gate $\rightarrow$ one-shot `Invoke()` $\rightarrow$ TCP [JSON](#) send.
- `src/filters.cpp`, `include/filters.h` — on-device filter/decimator implementations.
- `src/model_data.cc`, `include/norm_stats.h` — INT8 model blob; μ , σ and class names.
- `src/tcp_client.cpp`, `include/tcp_client.h` — length-prefixed [JSON](#) client to the RPi.
- `lib/FDC1004/fdc1004.*` — FDC1004 driver.
- `testing/Dataset/*` — make dataset, train, convert to INT8, emit `model_data.cc`.
- `testing/record_data.py`, `testing/gesture_display.py` — capture and quick plots.

Toolchain

PlatformIO [\[v\]](#), Arduino core/ESP-IDF [\[v\]](#), Python [\[v\]](#), TensorFlow/Keras [\[v\]](#).

```
# ESP32 firmware
pio run -e esp32dev -t upload
# Raspberry Pi process
cd raspi/GestureP && make && sudo systemctl restart gesturep
```

4.2.2 Sensor Sampling and Processing**4.2.2.1 Signal Acquisition**

In this section, the acquisition of raw capacitive data is explained in depth. The sampling cadence, channel synchronization, and calibration procedure are defined and described for both real-time inference and dataset recording.

The acquisition rate was set to 100 Hz. This is the lowest rate available in the FDC1004 configuration and is more than sufficient to detect even fast gesture motions.

Sampling cadence and timing — The firmware schedules sensor reads at a nominal sampling frequency of $f_s = 100$ Hz using a 10 ms tick driven by `micros()`. Each tick performs a *single* batched read of all four channels within the same loop iteration to maintain time alignment. The FDC1004 is configured in repeated-measurement mode so that conversions are available at each tick boundary.

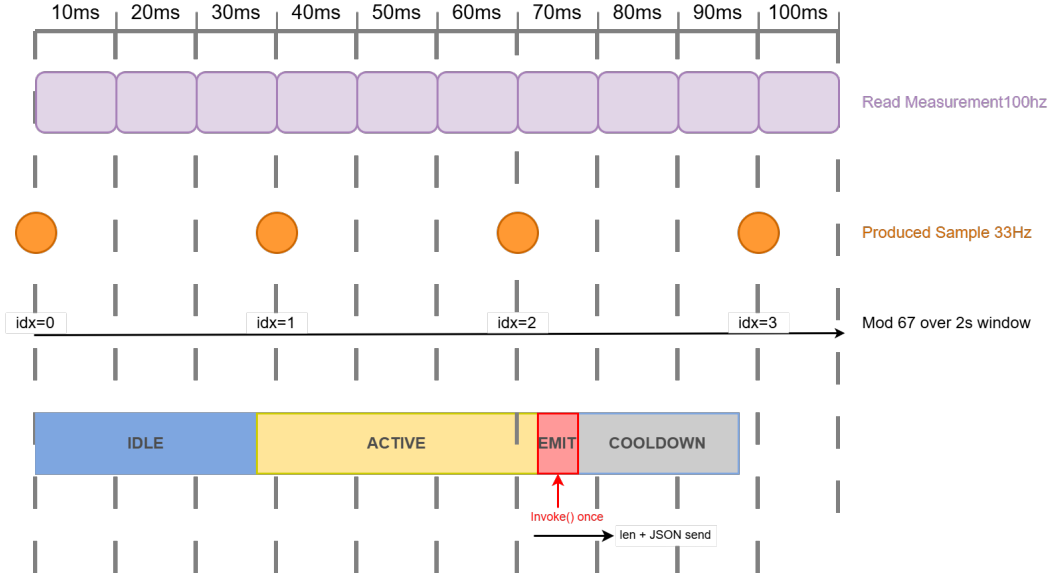


Figure 31: Timing diagram of the firmware sampling and processing pipeline.

Figure 31 illustrates the temporal alignment between sensor acquisition, preprocessing, and inference. All four channels are sampled synchronously at each firmware tick, after which decimation and buffering are applied to produce fixed-length windows for inference. The state-machine transitions governing inference invocation are also indicated, highlighting the point at which a single inference result is emitted and transmitted.

Details regarding decimation are addressed in the following Section 4.2.2.2.

Calibration and baseline — At boot, a short idle interval is used to (i) perform FDC1004 CAPDAC calibration per channel and (ii) estimate a per-channel baseline $\hat{x}_0$ by averaging initial samples. This baseline is subtracted from subsequent measurements prior to digital filtering.

Channel alignment — To guarantee supervision-ready data groups, a decimated output sample is emitted only when *all* channels have produced their downsampled value during the same tick (see Sections 4.2.2.2–4.2.2.2); otherwise, the frame is discarded. This ensures that the training tensor remains strictly chronological and aligned across channels.

Timestamps and logging format — Each raw and decimated sample vector is tagged with `t_us` (derived from `micros()`) and printed during recording as:

`t_us, ch0, ch1, ch2, ch3, label`

where `label` is one of {up, down, left, right, none}. This stream constitutes the sole input to the dataset builder.

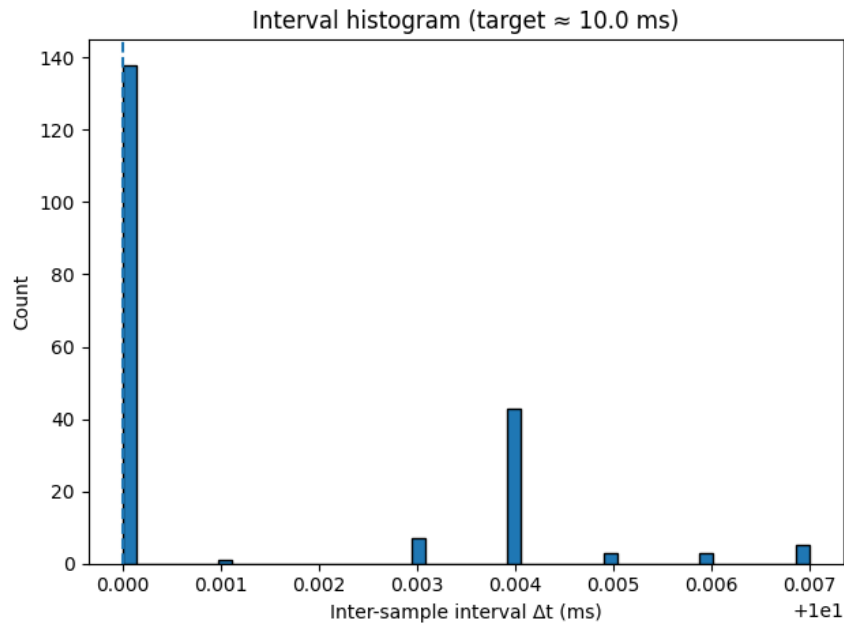


Figure 32: Histogram of inter-sample intervals

These values represent Δt computed from t_{us} over a 60 s run. The mean, p_{95} , and maximum are reported to characterize jitter around the 10 ms target.

Timing summary (100 Hz): mean=10.001 ms, p_{95} =10.005 ms, max=10.007 ms.

Minimal code excerpt (inline). The following excerpt illustrates only the scheduling and batched read logic; the full source is provided in the Appendix.

```
// 10 ms pacing at 100 Hz
static uint32_t lastRead = 0;
uint32_t now = micros();
if (now - lastRead >= 10000) {
    lastRead = now;
    FDC.readAllMeasurements(capacitance); // ch0..ch3
    // baseline subtract; push through conditioning/decimator
    // if produced -> tag with now and (optionally) print/log
}
```

Table 7: Acquisition parameters (fixed in firmware).

Parameter	Value	Notes
I ² C clock	400 kHz	ESP32 ↔ FDC1004
Raw sampling f_s	100 Hz	10 ms tick via <code>micros()</code>
Channels	4	single pass per tick
Timestamp	<code>t_us</code>	from <code>micros()</code>
Baseline	mean of idle window	subtracted before DSP
Log format	CSV	<code>t_us</code> , <code>ch0–ch3</code> , <code>label</code>

In short, the snippet enforces the 10 ms tick from Table 7, reads all four channels once per tick, timestamps with `micros()`, and applies baseline subtraction before passing samples to the conditioning/decimator. When logging is enabled, each tick is emitted as a CSV row `t_us`, `ch0..ch3`, `label`. This keeps samples time-aligned and ready for the downstream pipeline.

4.2.2.2 Signal Conditioning and Digital Filtering

This section documents the end-to-end signal conditioning pipeline used to prepare capacitive sensing data for gesture recognition. The approach is intentionally described with transparency: the final prototype relied on digital signal processing in firmware due to the absence of dedicated analog conditioning in the early hardware. The trade-offs of this design choice and the path toward a more mature analog front-end are explicitly stated.

Raw Signal Characterization and Spectral Analysis

The sensing system acquires four capacitive channels at a nominal sampling rate of

$$f_s = 100 \text{ Hz}. \quad (7)$$

Figure 33 shows a representative 60 s baseline-subtracted recording containing gesture activity. The signals exhibit (i) channel-dependent offsets and sensitivities, (ii) slow baseline variations, and (iii) transient gesture signatures.

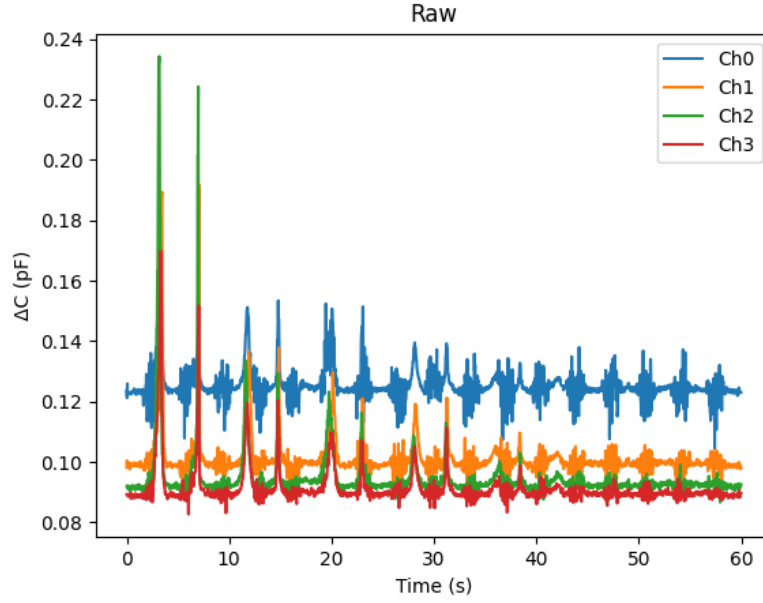


Figure 33: Baseline-subtracted raw capacitive signal over 60 s.

To quantify the frequency content and guide filter selection, the PSD was estimated using Welch's method over the same recording. Figure 34 shows that the dominant signal energy is concentrated below approximately 5 Hz, with a monotonic decay toward higher frequencies. No dominant narrowband interference (e.g., strong 50 Hz/60 Hz components) was observed in this dataset, although the spectral floor remains non-zero due to broadband sensor noise.

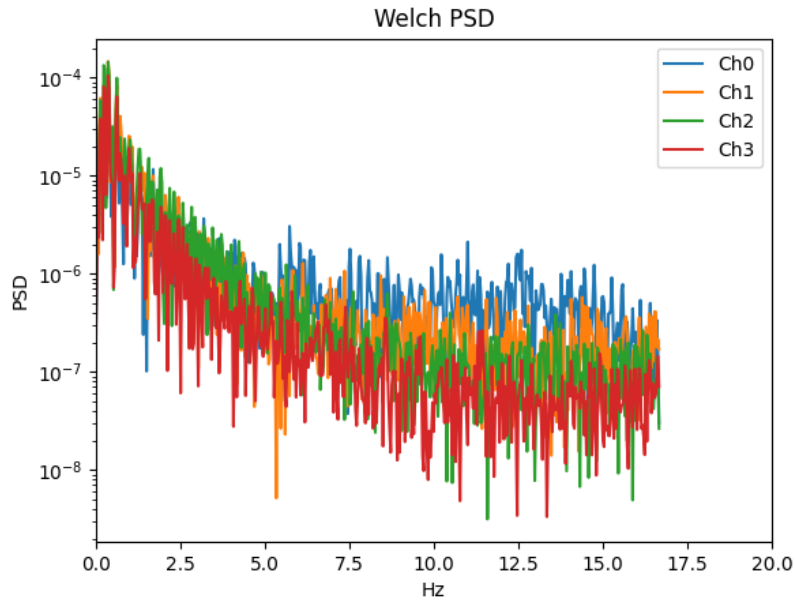


Figure 34: Welch PSD of the raw capacitive signal.

Design implication. The PSD observation that gestures occupy primarily the 0 Hz to 5 Hz band provides sufficient reasons for (a) applying a low-pass near 4.5 Hz, and (b) reducing the effective sampling rate via decimation while preserving gesture-discriminative information.

Overview of the Digital Processing Pipeline

The implemented pipeline, applied independently to each channel can be summarized as:

1. baseline subtraction (per-channel offset removal),
2. anti-aliasing low-pass filtering at the native sampling rate,
3. decimation (rate reduction),
4. high-pass filtering for drift mitigation,
5. post-decimation low-pass filtering for gesture-band smoothing.

Baseline Subtraction

A per-channel baseline is estimated at startup by averaging an initial set of samples while the sensor is idle. Denoting the raw capacitance by $x[n]$ and the baseline estimate by $\hat{x}_0$, the baseline-subtracted signal is:

$$x_{\Delta}[n] = x[n] - \hat{x}_0. \quad (8)$$

This step is effective in removing the offset introduced by prototype building variations, electrode geometry, and channel-dependent parasitics. Notably, it does not solve *time-varying* drift; this is addressed later in the pipeline.

Anti-aliasing Low-pass Filtering (Pre-decimation)

Because the processing pipeline reduces sampling rate, it is necessary to attenuate frequency components that could alias into the gesture band after downsampling. This is achieved with a low-pass filter applied at the native sampling rate $f_s = 100$ Hz prior to decimation. Based on PSD observations, an anti-alias cutoff of approximately

$$f_{c,\text{pre}} \approx 9 \text{ Hz} \quad (9)$$

was selected to preserve all gesture-relevant content (primarily below 5 Hz) while attenuating higher-frequency noise.

In implementation, this stage was realized as a stable second-order IIR (biquad) low-pass filter (Butterworth response), which provides a sharper transition band than a simple moving average and improves alias suppression.

Decimation — To reduce computational load and data throughput, the filtered stream is downsampled by an integer factor N . With

$$N = 3, \quad (10)$$

the effective sampling rate becomes:

$$f_{s,\text{eff}} = \frac{f_s}{N} = \frac{100 \text{ Hz}}{3} \approx 33.3 \text{ Hz}. \quad (11)$$

This rate remains comfortably above the gesture bandwidth, ensuring adequate temporal resolution and a Nyquist frequency of approximately 16.7 Hz.

Implementation note (consistency across channels) — Decimated samples are produced only when each channel has accumulated the required input samples, ensuring that the output vectors remain time-aligned across channels. This property is important for supervised learning, where the model assumes consistent sampling across input dimensions.

High-pass Filtering (Drift Mitigation)

Capacitive sensing is sensitive to slow environmental variation, including proximity changes, humidity, temperature, and supply/reference drift. These effects appear as low-frequency baseline motion that can reduce feature separability if left unmitigated. A high-pass (HP) filter is used to suppress this near-DC drift while retaining gesture dynamics. A first-order high-pass filter was implemented on the decimated stream, using the standard RC discretization:

$$\alpha_{\text{HP}} = \frac{RC}{RC + \Delta t}, \quad RC = \frac{1}{2\pi f_{c,\text{HP}}}, \quad \Delta t = \frac{1}{f_{s,\text{eff}}}, \quad (12)$$

$$y[n] = \alpha_{\text{HP}} (y[n-1] + x[n] - x[n-1]), \quad (13)$$

where $x[n]$ is the decimated signal and $y[n]$ is the HP output. Based on experimentation, a cutoff in the range

$$f_{c,\text{HP}} \in 0.2 \text{ Hz to } 0.5 \text{ Hz} \quad (14)$$

was explored. The objective was to remove slow drift while minimally distorting gesture content.

Observed limitation: cyclic baseline modulation — Across multiple recordings, a slow cyclic modulation of the noise floor and baseline was observed already in the raw signal. The HP filter does not create this phenomenon; however, it can *exaggerate* it by suppressing DC while re-weighting low-frequency components near the cutoff. In practice, this effect manifested as periods of increased apparent noise and occasional spikes after HP filtering. Despite this limitation, the HP stage was retained in the final configuration because it improved centering and consistency across sessions for gesture classification, and it reduced very slow baseline wander that otherwise propagated into the feature space.

Post-decimation Low-pass Filtering (Gesture-band Smoothing)

After drift mitigation, a low-pass filter is applied at $f_{s,\text{eff}} \approx 33.3$ Hz to suppress remaining broadband noise while preserving gesture morphology. PSD analysis of gesture recordings indicated that most useful energy lies below 5 Hz; therefore, the post-filter cutoff was selected as:

$$f_{c,\text{post}} \approx 4.5 \text{ Hz.} \quad (15)$$

Two low-pass realizations were considered:

1. First-order IIR (exponential moving average, EMA):

$$y[n] = \alpha_{\text{LP}} y[n-1] + (1 - \alpha_{\text{LP}}) x[n], \quad \alpha_{\text{LP}} = \exp\left(-2\pi \frac{f_{c,\text{post}}}{f_{s,\text{eff}}}\right). \quad (16)$$

2. Second-order Butterworth (biquad): offering a sharper roll-off and stronger stopband attenuation than the EMA filter.

The EMA approach is simple and computationally lightweight; the biquad provides improved attenuation of residual noise above the gesture band. In the final implementation, post-LP filtering was configured to achieve stable suppression of noise above approximately 5 Hz while maintaining clear transient gesture peaks.

Raw vs Filtered Gesture Signature

After defining the filtering implemented for signal conditioning, the comparison between gesture signatures before and after applying these filters:

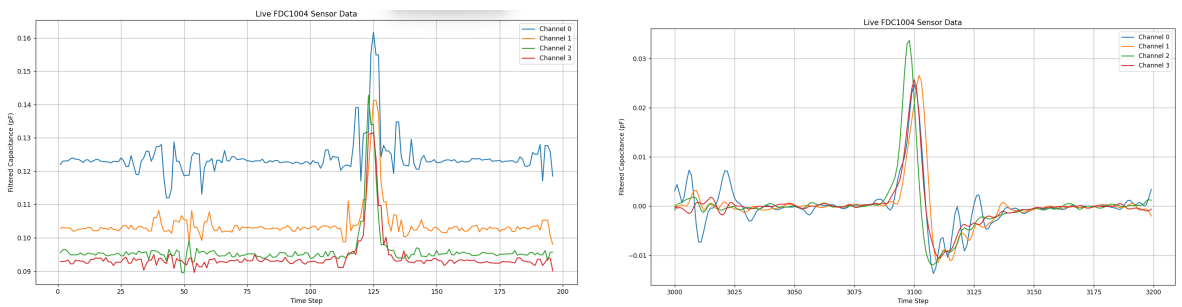


Figure 35: Comparison between Gesture Signatures, at the same distance (2cm). Left: Raw Signal; Right: Filtered Signal

Figure 35 illustrates the effect of the applied digital filtering on a representative gesture segment. Compared to the raw signal, the filtered output exhibits a substantially reduced noise floor while preserving the temporal structure and relative amplitude relationships of the gesture across channels. Transient gesture

features remain clearly identifiable, and no spurious oscillations or ringing are introduced by the filtering stages. This confirms that the selected combination of low-pass filtering, decimation, and drift mitigation improves signal quality without distorting gesture morphology.

Rationale for Keeping Digital Processing in the Prototype

The current prototype emphasizes rapid iteration and data collection for machine learning. Consequently, most conditioning was implemented digitally in firmware rather than via an analog front-end. This decision has a few advantages and known, important drawbacks:

Advantages —

- **Configurability:** cutoffs and decimation factors can be tuned quickly using [PSD](#) evidence without redesigning hardware.
- **Reproducibility:** identical digital filters can be applied consistently across datasets and experiments.
- **Data collection:** decimation reduces bandwidth and storage while preserving the gesture band.

Limitations —

- **Incomplete alias protection:** digital anti-aliasing is applied after measurement; strong high-frequency interference may already contaminate the sampled signal.
- **ADC dynamic range usage:** baseline drift consumes headroom before digital correction, reducing effective resolution.
- **Sensitivity to slow modulation:** cyclic baseline/noise modulation present in raw data can be emphasized by low-cutoff HP filtering (Section [4.2.2.2](#)).
- **Computational load and latency:** the combination of recursive filtering, decimation buffering, and per-channel state updates introduces non-zero algorithmic latency and consumes a significant portion of the available processing budget. As a result, increasing the acquisition rate beyond the chosen 100 Hz was avoided, as higher rates exacerbated noise and timing instability under the constraints of the prototype hardware.
- **Phase and temporal distortion:** recursive filters introduce frequency-dependent phase delay; while acceptable for offline training, this must be considered in real-time gesture segmentation.

Planned Hardware Improvements

The reliance on digital filtering can be accepted for the current proof-of-concept, but a more mature design would include dedicated analog conditioning in a revised PCB. Planned improvements include:

- **Analog anti-alias filtering** before digitization to suppress high-frequency interference that cannot be reliably removed after sampling.
- **Improved reference and supply conditioning** (low-noise regulators, reference decoupling, layout practices) to reduce cyclic baseline modulation.

These hardware changes can be expected to reduce the amplitude of baseline drift and cyclic modulation at the source, thereby decreasing the need for aggressive low-cutoff digital high-pass filtering and improving the signal-to-noise ratio of gesture signatures.

4.2.2.3 Summary of Final Parameters

Table 8 summarizes the final configuration used for data collection.

Table 8: DSP configuration used for gesture data collection.

Stage	Parameter	Value
Sampling	f_s	100 Hz
Pre-LP (anti-alias)	$f_{c,\text{pre}}$	9 Hz
Decimation	N	3
Effective rate	$f_{s,\text{eff}}$	33.3 Hz
High-pass (drift)	$f_{c,\text{HP}}$	0.2 Hz to 0.5 Hz
Post-LP (gesture)	$f_{c,\text{post}}$	4.5 Hz

4.2.3 Preprocessing for Machine Learning

4.2.3.1 Overview and Goals

Flow and artefacts

Purpose — Convert each labeled recording into a fixed-size tensor $[67, 4]$ in chronological order, then standardize per channel using training-only statistics. This guarantees training–inference parity with the firmware normalization constants embedded in `norm_stats.h`.

Table 9: Preprocessing artefacts.

File	Contents
<code>dataset_67x4.npz</code>	$X_{\text{tr}}, X_{\text{va}}, X_{\text{te}} \in \mathbb{R}^{N \times 67 \times 4}; y(\cdot) \in \{0, \dots, 4\}$
<code>norm_stats.h</code>	Per-channel μ, σ and fixed class list (firmware parity)

Outputs —**4.2.3.2 Windowing and Time Normalization****Resampling to 67 points**

Uniform resampling — For each CSV, the time column `t_us` defines the sample span $[t_{\min}, t_{\max}]$. A uniform grid of exactly $K = 67$ timestamps is generated across this span, and each of the four channels is linearly interpolated to that grid:

$$t^*(i) = t_{\min} + \frac{i}{66} (t_{\max} - t_{\min}), \quad x^*(i, c) = \text{interp}(t^*(i); (t, x_c)), \quad i = 0..66, \quad c = 0..3.$$

This yields a single $[67, 4]$ window per recording, preserving temporal order and equalizing sample counts across the corpus (ref.: `make_dataset.py`).

Rationale — This step removes minor cadence variability between recordings and matches the firmware's $2\text{ s} / 67\text{-step}$ inference window, ensuring the model sees a consistent temporal resolution.

4.2.3.3 Standardization and Label Encoding**Channel-wise standardization**

Training-only statistics — Let $X_{\text{tr}} \in \mathbb{R}^{N_{\text{tr}} \times 67 \times 4}$. Per-channel statistics are computed on the training split only:

$$\mu_c = \frac{1}{N_{\text{tr}} \cdot 67} \sum_{n,t} X_{\text{tr}}[n, t, c], \quad \sigma_c = \sqrt{\frac{1}{N_{\text{tr}} \cdot 67} \sum_{n,t} (X_{\text{tr}}[n, t, c] - \mu_c)^2 + \varepsilon}.$$

Each split is standardized as $Z = (X - \mu) / \sigma$. The resulting μ, σ are exported verbatim to firmware (see below).

Firmware parity — The compiled constants match `norm_stats.h`:

$$\mu = [0.00022758, 0.00027420, 0.00025576, 0.00025231],$$

$$\sigma = [0.00820502, 0.00856527, 0.00765611, 0.00709972].$$

These are used on-device to normalize each channel before quantization, guaranteeing parity with training.

Label schema

Class order — Fixed mapping (used in training, firmware, and [JSON](#) emission):

$$\text{up} \rightarrow 0, \text{down} \rightarrow 1, \text{left} \rightarrow 2, \text{right} \rightarrow 3, \text{none} \rightarrow 4.$$

This is consistent with `CLASSES` in `norm_stats.h` and `make_dataset.py`.

4.2.3.4 Splits, Balance, and Augmentation

Dataset split

Stratified split — The corpus is split stratifiably into train/validation/test with 70/15/15 proportion (two-stage split with `random_state=42` in `make_dataset.py`), preserving class balance across splits.

Light augmentation (training only)

Temporal roll — To improve shift robustness without breaking parity, mini-batches may be *circularly* shifted in time by a random integer in $[-m, +m]$ samples (default $m = 10$) with probability $p = 0.5$ (see `multiseedTrain.py`: `roll_batch` and `gen`). No amplitude jitter or noise is applied in the final setup.

4.2.3.5 Integrity and Reproducibility

Checks and artefacts

Sanity checks — Ensure chronological order; no NaN/Inf; expected per-class counts; standardized distributions with mean ≈ 0 , std ≈ 1 on training.

Exported artefacts

What is produced —

leftmargin=1.5em `dataset_67x4.npz`, containing arrays $X_{\text{tr}}, X_{\text{va}}, X_{\text{te}} \in \mathbb{R}^{N \times 67 \times 4}$ and $y_{\text{tr}}, y_{\text{va}}, y_{\text{te}} \in \{0, \dots, 4\}$, plus μ, σ , and the class list.

leftmargin=1.5em `norm_stats.h`, a C header exporting the same μ, σ and the class names for firmware.

4.2.4 Model Training, Quantization, and Embedded Deployment

4.2.4.1 Model and Training

Topology and Hyperparameters

Architecture — The classifier is a temporal CNN built from two Conv1D blocks followed by a lightweight pooling head. An input window of size $[67, 4]$ is processed by Conv1D(16, $k=9$, *same*, no bias), Batch Normalization, and ReLU. A second Conv1D(32, $k=9$, *same*, no bias) with Batch Normalization and ReLU increases the receptive field without a large parameter cost. We then apply both GlobalAveragePooling1D and GlobalMaxPooling1D and concatenate the results, which captures mean and peak evidence and makes the model robust to small time shifts. A Dropout layer ($p=0.2$) regularizes the head, followed by Dense(32, ReLU) and a Dense(5, Softmax) output.

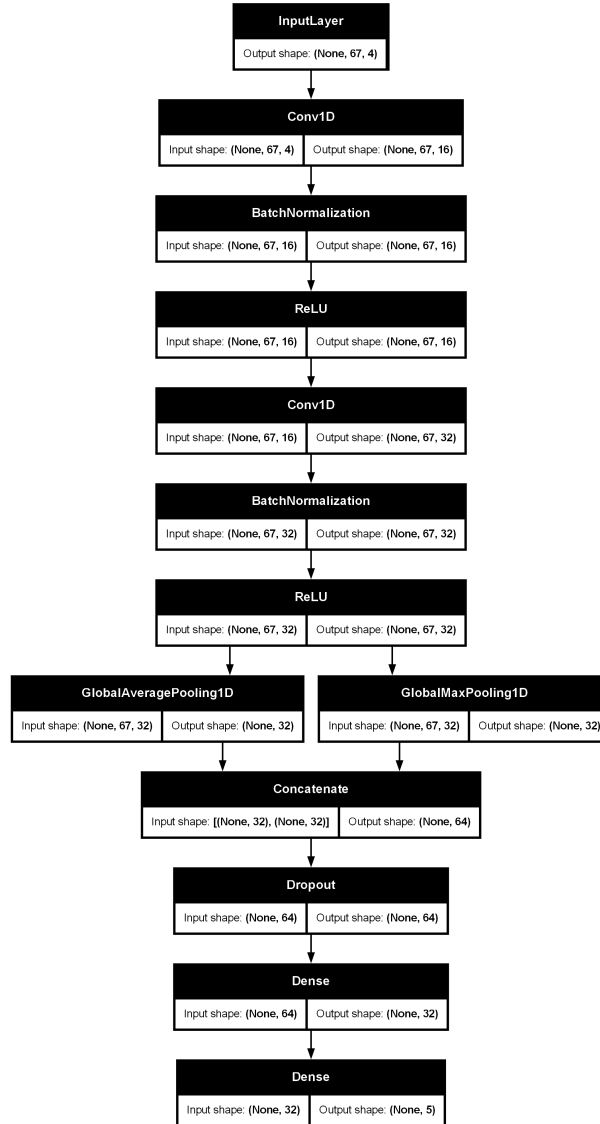


Figure 36: Model Architecture

Rationale — Gestures appear as characteristic temporal signatures across the four capacitive channels; a 1D CNN provides an appropriate inductive bias for learning these local temporal patterns and cross-channel interactions with low model complexity. Batch normalization improves training stability and reduces sensitivity to amplitude scaling and baseline differences common in capacitive sensing.

A kernel $k=9$ at ≈ 33 Hz spans ~ 270 ms of context; stacking two layers expands the receptive field without a serious cost. Global average *and* max pooling provide time-shift robustness and capture both salient transient events and overall gesture profile; concatenation aggregates mean and extreme evidence; dropout regularizes the head.

Training setup

We train with Adam (learning rate: 10^{-3}) and sparse categorical cross-entropy. Early stopping is driven by validation loss. To improve shift robustness while keeping parity with deployment, we apply random circular time rolls to training windows with probability 0.5 and a small maximum shift. No amplitude jitter or noise is used in the final runs. Figure 37 shows rapid convergence and stable validation behavior for the selected seed.

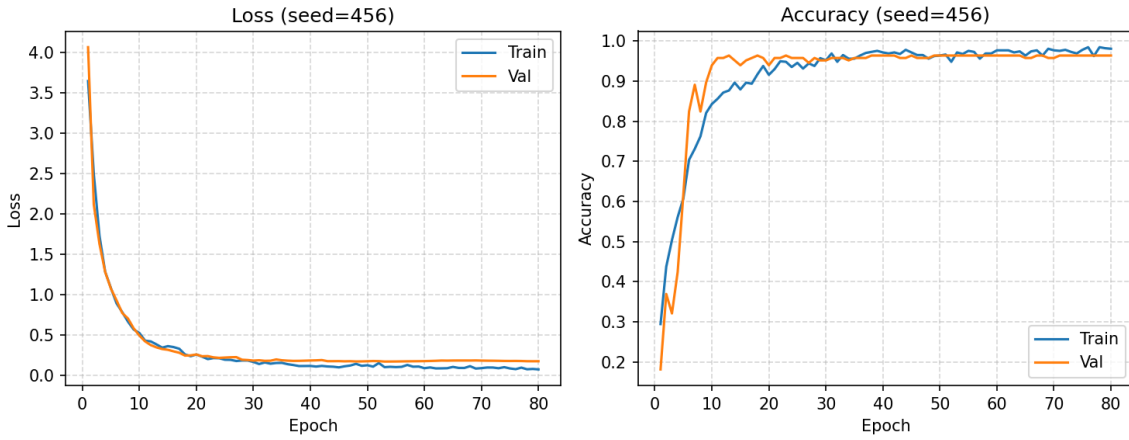


Figure 37: Learning curves for the selected seed: fast convergence, stable validation beyond early epochs (seed 456).

Multi-Seed Training and Selection

Protocol — To account for stochastic optimization, we train the same architecture and hyperparameters on identical splits/transforms across seeds $\{101, 123, 231, 345, 456, 567, 678, 789, 890, 999\}$. Augmentation (random circular time roll) is applied on training only.

Stability and choice — Across seeds, validation macro-F1 is 0.9575 ± 0.0055 (weighted-F1 0.9568 ± 0.0055), indicating low variance. We select the checkpoint with the highest validation macro-F1—seed **456** at 0.9643. Individual scores are reported in Table 10.

Table 10: Validation F1 across seeds. Dispersion is small; seed 456 is best.

Seed	Macro-F1	Weighted-F1
101	0.9642	0.9635
123	0.9462	0.9453
231	0.9585	0.9573
345	0.9506	0.9511
456	0.9643	0.9636
567	0.9518	0.9513
678	0.9627	0.9631
789	0.9587	0.9575
890	0.9587	0.9576
999	0.9585	0.9574
Mean $\pm$ std (macro-F1): 0.9575 ± 0.0055 ; weighted-F1: 0.9568 ± 0.0055		

Test metrics (seed 456)

Aggregate performance — On the held-out test set the model reaches a macro-F1 of 0.9811 and a weighted-F1 of 0.9818. Class-wise precision, recall, and F1 are listed in Table 11. The normalized confusion matrix (Fig. 38) shows errors are rare and mostly between *down/left* and *right/none*.

Table 11: Per-class test metrics for the selected seed.

Class	Precision	Recall	F1
up	1.0000	1.0000	1.0000
down	0.9667	0.9667	0.9667
left	0.9667	0.9667	0.9667
right	1.0000	0.9667	0.9831
none	0.9783	1.0000	0.9890

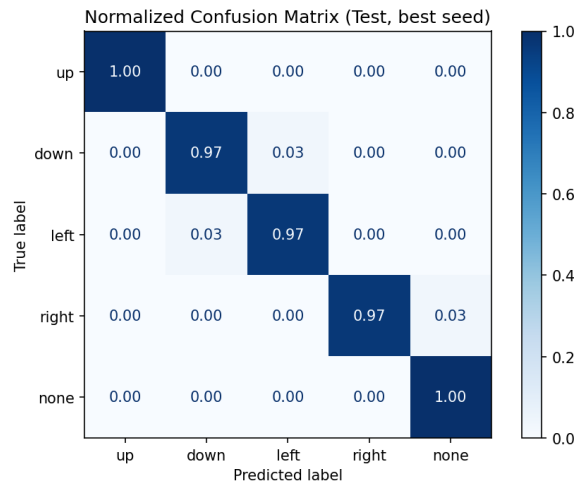


Figure 38: Normalized confusion matrix on the test set (seed 456). Off-diagonals are confined to $\{\text{down}, \text{left}\}$ and $\{\text{right}, \text{none}\}$ at $\sim 3\%$.

Temporal shift robustness — We also evaluated robustness to small timing shifts by circularly rolling each test window by $\{-12, -8, -4, 0, +4, +8, +12\}$ samples (about ± 360 ms at 33 Hz). Accuracy remains between 0.982 and 0.988 across these shifts, consistent with the use of global pooling.

Table 12: Accuracy under temporal rolls (test set, seed 456).

Shift (samples)	Accuracy
-12	0.988
-8	0.988
-4	0.988
0	0.982
+4	0.988
+8	0.982
+12	0.988

4.2.4.2 Quantization and TFLite Export

Representative dataset and converter

Calibration — Quantization uses a class-balanced subset of normalized $[67, 4]$ training windows as the representative dataset. We disable all augmentations during calibration so the activation ranges reflect what the firmware will produce.

Export — We perform full-integer quantization: inputs, weights, activations, and logits are INT8. Batch Normalization layers are folded into the preceding convolutions, and Dropout is removed at inference. The resulting flatbuffer (`gesture_int8.tflite`) is converted to a C array (`model_data.cc`) for PlatformIO.

Runtime operator set

Operators — The deployed model uses the following TFLite-Micro kernels:

- CONV_1D (or lowered CONV_2D)
- RELU
- REDUCE_MEAN, REDUCE_MAX
- CONCATENATION
- FULLY_CONNECTED
- SOFTMAX
- QUANTIZE, DEQUANTIZE

4.2.4.3 Embedded Deployment on ESP32

Interpreter setup and memory

Integration — At boot, the firmware creates a single `tflite::MicroInterpreter` using a static tensor arena of approximately 40 KB. The input tensor has shape `[1, 67, 4]` and INT8 type; the output is an INT8 vector of length five. Before each inference, the firmware applies the same channel-wise mean and standard deviation used during training (compiled into `norm_stats.h`) and then quantizes the normalized window with the model's input scale and zero-point.

```
Starting calibration...
Channel 0 → CAPDAC=0, Baseline=1.8016 pF
Channel 1 → CAPDAC=0, Baseline=1.8441 pF
Channel 2 → CAPDAC=0, Baseline=1.8986 pF
Channel 3 → CAPDAC=0, Baseline=2.0077 pF
Calibration complete.

1535
Commands:
  start_up / start_down / start_left / start_right / start_none
  stop
  infer_on / infer_off
ESP32_READY
TFLM input dims: 1 x 67 x 4
```

Figure 39: ESP32 boot and calibration.

On Figure 39 the process of setup calibration is printed at startup, CAPDAC/baseline are established and TFLM reports input tensor shape, confirming the on-device model is loaded correctly

```
ESP32_READY
TFLM input dims: 1 x 67 x 4
infer_on
Inference: ON
OUT scale=0.003906, zp=-128, ncls=5
PRED_EVT,right,conf=1.00, tcp_send=OK
PRED_EVT,up,conf=1.00, tcp_send=OK
PRED_EVT,right,conf=0.87, tcp_send=OK
PRED_EVT_IGNORED,left,conf=0.38
PRED_EVT,left,conf=0.81, tcp_send=OK
```

Figure 40: ESP32 runtime excerpt with multiple gestures.

Figure 40 shows the gate emitting multiple gesture, inference runs (`PRED,<class>`), and length-prefixed `JSON` frames are sent to the Raspberry Pi. There is also a case of ignored event due to low confidence in a particular gesture (performed too far away or too quickly).

Runtime path and decision

Emission — When the gesture gate signals an event, the device runs one inference and accepts the top-1 class if its (dequantized) softmax score exceeds the threshold τ (value set in firmware). It then sends

a length-prefixed [JSON](#) frame containing the class label, confidence, device identifier, and timestamp to the Raspberry Pi.

Footprint and latency

Measurements — We report end-to-end inference time (mean and 95th percentile), the time for input copy and quantization, the size of the model blob in `model_data.cc`, and the peak tensor-arena usage. These measurements allow reproduction on identical hardware and build settings.

4.2.5 Communication protocol to the Raspberry Pi

4.2.5.1 Overview

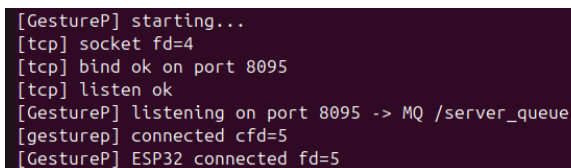
Role and scope

Purpose — The ESP32 acts as a TCP *client* that delivers gesture events to a Raspberry Pi *server* process (“GestureP”). The Pi process validates and maps each event to the gateway’s system queue.

4.2.5.2 Transport

Endpoint and session

Socket — A single long-lived TCP session is established from the ESP32 to the Raspberry Pi on port **8095**. The ESP32 reconnects on failure with a short back-off. The Pi listens on `INADDR_ANY` and accepts one or more clients.



```
[GestureP] starting...  
[tcp] socket fd=4  
[tcp] bind ok on port 8095  
[tcp] listen ok  
[GestureP] listening on port 8095 -> MQ /server_queue  
[gesturep] connected cfd=5  
[GestureP] ESP32 connected fd=5
```

Figure 41: GestureP server startup on the Raspberry Pi.

Keepalive — Standard TCP keepalive is enabled on the server side; the client relies on application traffic (gesture events) to keep the connection up.

4.2.5.3 Framing and Encoding

Length-prefix frames

Record structure — Each application frame consists of a 2-byte big-endian length header followed by an UTF-8 [JSON](#) payload of that length. Frames are sent back-to-back on the same stream; there are no delimiters or line breaks. The maximum payload length used in this work is well below 512 bytes.

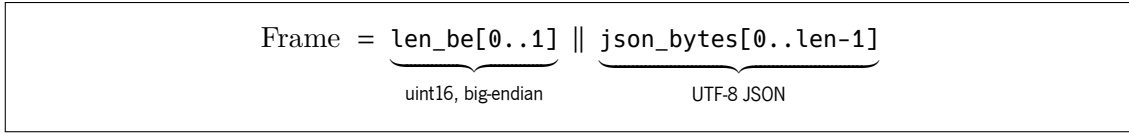


Figure 42: Application framing. The server first reads two bytes to obtain the [JSON](#) payload length, then reads exactly that many bytes.

4.2.5.4 Payload Schema

Gesture event

JSON fields — The ESP32 sends one [JSON](#) object per emitted gesture. The schema below matches the firmware and server implementation.

Table 13: Gesture event payload (ESP32 → Raspberry Pi).

Field	Type	Description
type	string	Literal "gesture_evt"
device_id	string	ESP32 identifier (e.g., "esp32-01")
ts_ms	integer	Millisecond timestamp from device
gesture	string	One of {up, down, left, right, none}
conf	number	Dequantized softmax score in [0, 1]

Example

```
len (2 bytes, BE) = 87
{"type": "gesture_evt", "device_id": "esp32-01", "ts_ms": 1712345678,
 "gesture": "up", "conf": 0.87}
```

4.2.5.5 Runtime Behavior

Client state machine

Connect–send–reconnect — After boot, the client resolves the server address, opens a TCP socket to port 8095, and switches to a streaming loop. When the gesture gate in firmware emits a single label, the client serializes one [JSON](#) payload, prefixes it with the 2-byte length, and writes the frame. If `send()` fails or the connection drops, the client closes the socket and retries after a short delay. This preserves at-most-once delivery semantics and avoids duplicate frames.

Server receive loop

Read contract — The server reads exactly two bytes for the header, interprets the length as big-endian, then *blocking-reads* until that many payload bytes are received. If the header or payload cannot be fully

read, the server treats the connection as broken and waits for a new client. Valid [JSON](#) is parsed and handed to the mapping layer.

```
[tcp] rcv 2 bytes (total 4/2)
[tcp] header len=82 (h0=0x00 h1=0x52)
[tcp] rcv 82 bytes (total 164/82)
[map] gesture=up conf=1.00 -> {"type":"panel_cnd","target":"ambient","cnd":"brightness_delta","value":5}
[msg] sent
[tcp] rcv 2 bytes (total 4/2)
[tcp] header len=85 (h0=0x00 h1=0x55)
[tcp] rcv 85 bytes (total 170/85)
[map] gesture=right conf=0.87 -> {"type":"panel_cnd","target":"ambient","cnd":"next_effect"}
[msg] sent
[tcp] rcv 2 bytes (total 4/2)
[tcp] header len=84 (h0=0x00 h1=0x54)
[tcp] rcv 84 bytes (total 168/84)
[map] gesture=left conf=0.81 -> {"type":"panel_cnd","target":"ambient","cnd":"prev_effect"}
```

Figure 43: Server receive loop in operation.

Figure 43 shows this process. The commands received are from the sessions presented in figure-40. Note that the ignored event was not sent and the other gestures went through cleanly. The print also shows the sending of the command via message queue.

Contract in one paragraph

Protocol — The ESP32 maintains a persistent TCP client connection to the Raspberry Pi on port 8095. Each gesture event is encoded as a single UTF-8 [JSON](#) object with fields {type, device_id, ts_ms, gesture, conf} and transmitted using a 2-byte big-endian length prefix. The server reads the header, consumes the exact number of bytes, parses the [JSON](#), and forwards the command into the local control pipeline. This minimalist contract is sufficient for reliable, low-overhead delivery and matches the deployed firmware and server implementations.

Chapter Summary

This chapter presented the concrete implementation of the proposed capacitive gesture recognition system as a fully integrated hardware–software prototype. The sensing module was realized as a two-layer physical structure comprising a custom front-panel electrode array with active shielding and a back-panel electronics assembly based on an ESP32 microcontroller and an FDC1004 capacitive sensing IC. The mechanical enclosure and mounting system were entirely 3D-printed, enabling reproducible geometry, shielding integrity, and flexible deployment for testing. On the software side, the complete embedded firmware stack was implemented on the ESP32, including high-rate capacitive data acquisition, signal preprocessing, temporal buffering, and on-device inference using a quantized machine learning model. Supporting tooling for data capture, dataset construction, training, and model conversion was developed in Python and integrated into a unified workflow. The gesture recognition module was further embedded into a larger prototype system via TCP-based communication with a Raspberry Pi server, enabling end-to-end operation and external actuation. Together, these elements demonstrate the practical feasibility, robustness, and integration readiness of the proposed gesture recognition approach.

Chapter 5

Verification and Results

This chapter's aim is to establish all tests the developed system went through, document the results and establish the appropriate conclusions. These tests will validate the complete gesture-recognition system from sensors and acquisition timing through preprocessing parity, classifier performance, embedded runtime, and network communication. Wherever possible, there will be both functional correctness and quantitative performance documented.

5.1 Sensor and Front-End Characterization

This section quantifies the intrinsic signal quality of the four-channel capacitive proximity sensing front-end using a truly raw acquisition (i.e., prior to any baseline removal, detrending, or other digital conditioning). The objective is to characterize idle behavior in terms of short-term noise, long-term drift, and the resulting minimum detectable capacitance change.

5.1.1 Idle Noise, Drift, and Minimum Detectable Change

A 120 s idle recording was acquired at 100 Hz with the system at rest and no user interaction. Figure 44 shows the raw time-domain outputs of all four channels over the full acquisition window. This plot provides qualitative context for the metrics reported in Table 14, highlighting (i) the relative noise levels across channels, (ii) the static operating offsets, and (iii) low-frequency baseline evolution observable as drift.

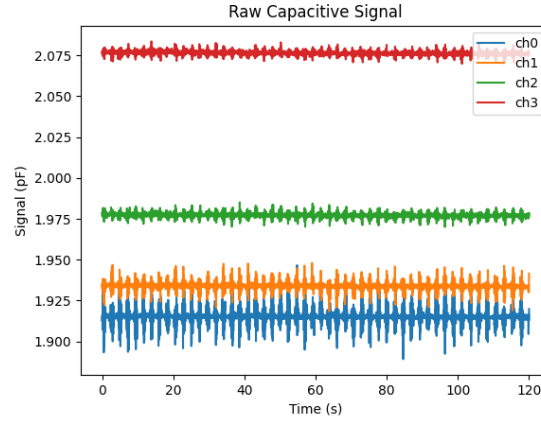


Figure 44: Raw time-domain output of the four capacitive channels during a 120 s idle acquisition at 100 Hz, prior to any baseline removal or digital conditioning.

For each channel, the RMS noise was computed after mean removal over the full acquisition window, thereby quantifying signal variability around the operating point. Long-term drift was estimated via a least-squares linear fit to the raw signal and is reported in pF/min . The front-end quantization step was fixed for all channels at $1.0 \times 10^{-4} \text{ pF}/\text{LSB}$ in this configuration; because it is common across channels, it is reported here in text rather than as a per-channel table entry.

Finally, a noise-limited resolution metric is reported as the minimum detectable capacitance change under a 1σ criterion. Under this definition, the minimum detectable change equals the measured RMS noise (in pF_{RMS}) and represents the practical sensitivity limit imposed by the combined front-end and environmental noise during idle operation.

Table 14: Per-channel idle noise, drift, and noise-limited resolution from raw data at 100 Hz (duration 120 s).

Channel	Noise (RMS, pF)	Drift (pF/min)	Min. Detectable Change (pF_{RMS})
ch0	5.335×10^{-3}	-7.70×10^{-5}	5.335×10^{-3}
ch1	2.894×10^{-3}	-2.99×10^{-4}	2.894×10^{-3}
ch2	1.627×10^{-3}	-3.31×10^{-4}	1.627×10^{-3}
ch3	1.464×10^{-3}	-5.18×10^{-4}	1.464×10^{-3}

Across channels, ch0 exhibits the highest idle RMS noise, while ch2 and ch3 present the lowest. Drift remains small over the 120 s window but is consistently negative, indicating a slow baseline decrease that should be accounted for in higher-level processing (e.g., baseline tracking) during extended operation. In the following section, the frequency-domain analysis further disambiguates broadband noise from structured components and low-frequency contributions.

Idle vs Gesture PSD

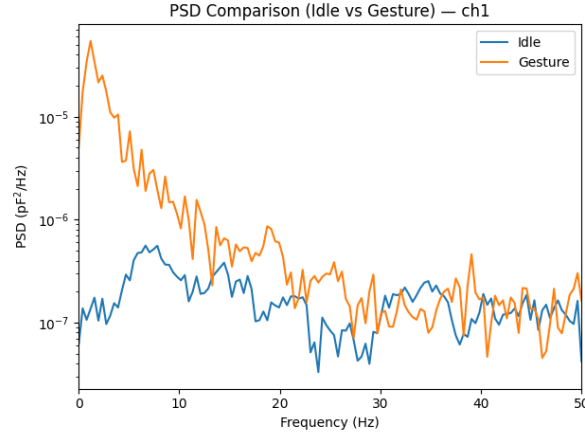


Figure 45: PSD of a representative capacitive channel under idle conditions and during a representative gesture, computed using Welch’s method (Hann window, 256-sample segments, 50% overlap).

Figure 45 compares the power spectral density of one capacitive channel under idle conditions and during a representative gesture, the other channel behave similarly. Gesture activity results in a pronounced increase in spectral energy at low frequencies, with the strongest separation observed below approximately 5 Hz. In contrast, the high-frequency noise floor above 20 Hz remains comparable between idle and gesture conditions, indicating that gesture-induced variations are primarily low-frequency in nature and do not introduce additional broadband electrical noise. These observations support the use of low-pass or band-limited processing for gesture detection.

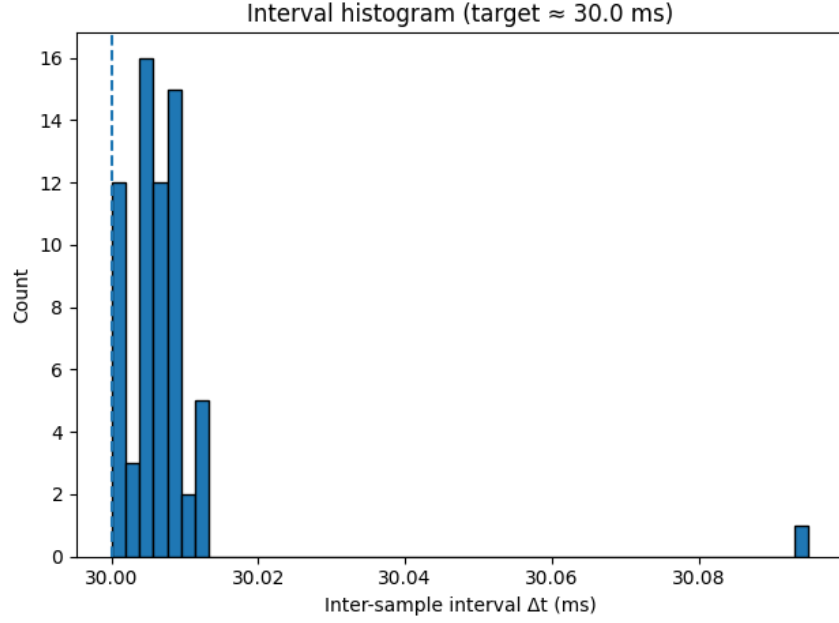
5.2 Acquisition Timing

5.2.1 Sampling Cadence at 100 Hz (Raw)

The raw inter-sample interval characterization was presented previously in Chapter 4 (see Fig. 32). We refer to that result here and note that it satisfies the timing budget for the downstream pipeline. The values obtained: mean=10.001 ms, p95=10.005 ms, max=10.007 ms.

5.2.2 Post-Pipeline Emission Cadence at 33 Hz

After filtering and decimation, the emission interval targets 30.0 ms. Figure 46 shows the measured histogram and table 15 summarizes the distribution. The observed dispersion is minimal (sub-0.1 ms), indicating that the pipeline preserves a stable cadence.

Figure 46: Inter-emit interval histogram after the full pipeline (≈ 33 Hz).Table 15: Inter-emit interval statistics at ≈ 33 Hz. Nominal target ≈ 30.0 ms.

	Mean (ms)	p95 (ms)	p99 (ms)	Max (ms)
Measured	30.007	30.012	30.041	30.095

The post-pipeline cadence tracks the nominal 30.0 ms period with sub-0.1 ms jitter (p99–mean ≈ 0.034 ms), which is well within the real-time budget and ensures consistent temporal alignment for the $[67 \times 4]$ inference windows.

5.3 Impact of the Filtering Pipeline

Figure 47 illustrates how a representative swipe gesture (channel 1) evolves as it passes through the successive stages of the signal processing pipeline. The input signal shown at the top corresponds to the baseline-stabilized acquisition stream used during normal operation, where a constant offset is removed at start-up.

The first processing stage applies a pre-low-pass filter with a cutoff of 9 Hz at the original 100 Hz sampling rate. This step visibly smooths the input signal and suppresses high-frequency fluctuations while preserving the overall gesture shape. Its primary role is to limit the signal bandwidth prior to downsampling, reducing the risk of aliasing in the subsequent decimation stage.

Decimation by a factor of three reduces the sampling rate to approximately 33.3 Hz and further smooths the signal through implicit averaging. At this point, the main temporal structure of the gesture is clearly

retained, but short-term noise is significantly reduced. The high-pass stage that follows (0.5 Hz cutoff) removes slow baseline components and long-term decay, effectively isolating the dynamic portion of the gesture. Finally, a post-low-pass filter with a cutoff of 4.5 Hz produces a clean and stable output waveform suitable for downstream processing.

One clear drawback of the full processing chain is also visible in Fig. 47: the gesture amplitude at the final output is noticeably lower than the initial peak observed at the input. This reduction is expected given the cumulative effect of low-pass filtering, decimation averaging, and high-pass filtering, but it highlights the trade-off between noise suppression and signal amplitude. In this implementation, the pipeline prioritizes signal stability and robustness over preserving absolute peak magnitude.

In practice, this attenuation is not inherently problematic, as gesture detection relies primarily on the temporal shape and relative variation of the signal rather than its absolute amplitude. Nevertheless, future versions of the system could mitigate this drawback by moving part of the filtering chain into hardware. Hardware-implemented filtering would reduce the need for aggressive digital processing, preserve more of the original signal amplitude, and simplify the downstream signal conditioning stages.

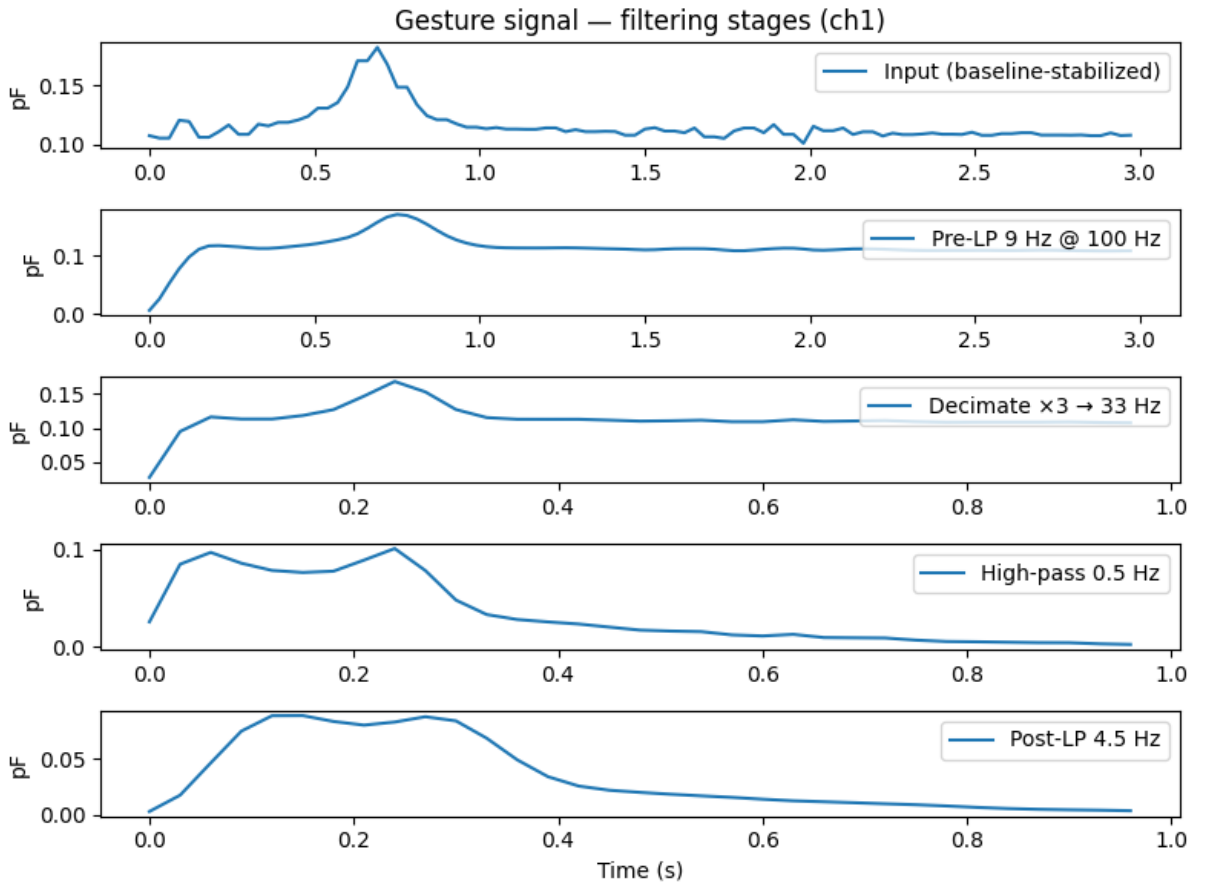


Figure 47: Evolution of a representative swipe gesture (channel 1) through successive processing stages, from baseline-stabilized input to final filtered output.

5.4 Classifier Performance — Summary

The detailed model architecture, training setup, seed sweep, and full confusion analysis are presented in Section 4.2.4 of the Implementation chapter (see Table 10, Fig. 37, and Fig. 38). Here we summarize the headline results for completeness and to support comparisons with on-device and end-to-end evaluations reported later in this chapter.

Table 16: Offline headline metrics (held-out test set; seed selected in Table 10).

Metric	Value	Notes
Macro-F1	0.9811	Best seed (456)
Weighted-F1	0.9818	
Per-class F1	up: 1.000; down: 0.967; left: 0.967; right: 0.983; none: 0.989	From Fig. 38/Table 11
Seed stability (val)	0.9575 ± 0.0055 (macro-F1)	Table 10
Temporal shift robustness	0.982–0.988 accuracy (± 12 samples)	Table 12

As discussed in Section 4.2.4, errors are rare and primarily confined to the *down/left* and *right/none* pairs, consistent with the physical similarity of those trajectories in certain contexts (Fig. 38). The validation seed sweep (Table 10) shows low variance across random initializations, and the global pooling head confers robustness to timing shifts (Table 12).

5.5 Classifier Performance (Offline)

We summarize training stability, test-set performance, and temporal shift robustness. The validation seed sweep shows low variance; the best seed achieves macro-F1 []. The normalized confusion matrix indicates rare confusions consistent with deployment behavior.

5.5.1 Embedded Runtime (ESP32)

This subsection quantifies the deployment footprint and runtime performance of the selected INT8 model when executed on the ESP32 using TensorFlow Lite for Microcontrollers (TFLM). Measurements were collected on-device by timestamping preprocessing and inference calls with microsecond resolution.

5.5.1.1 Footprint

Table 17 reports the model flash size (the compiled `.tflite` array length) and the configured tensor arena allocation. We also report the observed heap delta measured around arena/interpreter setup, which provides a coarse indication of additional runtime overhead beyond the statically allocated arena.

Table 17: Embedded deployment footprint on ESP32 (TFLM INT8).

Metric	Value
Model size (Flash)	15,656 bytes
Tensor arena (allocated)	61,440 bytes
Observed heap delta after allocation	196 bytes

5.5.1.2 Inference latency

Latency was measured by timing (i) the input preparation step (normalization and quantization into the input tensor) and (ii) the TFLM `Invoke()` call. Across the collected runs, preprocessing remained well below 1 ms (typically ~ 0.26 ms), while `Invoke()` was tightly clustered around ~ 25 ms. Table 18 summarizes these results.

Table 18: On-device runtime latency on ESP32 (TFLM INT8).

Metric	Latency
Preprocessing time (typ.)	~ 0.26 ms
<code>Invoke()</code> time (typ.)	~ 24.8 ms

Overall, the measured footprint and latency indicate that the deployed model executes comfortably within the memory constraints of the ESP32 and supports real-time inference at the effective sampling rate used by the gesture pipeline.

5.6 Field Test by Distance

A field test was conducted to evaluate robustness as a function of hand–sensor distance. For each distance in $\{1, 2, 3, 4, 5\}$ cm, five trials were performed per gesture class (*left*, *right*, *up*, *down*). A trial is counted as a *valid detection* when the expected gesture label is emitted as a `PRED_EVT`. Trials that produced either a different label or no accepted label were counted as *non-detections*. Table 19 reports the resulting per-class detection rates; Table 20 reports the mean confidence of valid detections only (i.e., conditioning on successful detections).

Detection Rate vs Distance

As expected for capacitive proximity sensing, performance degrades with increasing distance as the gesture-induced signal amplitude approaches the idle noise floor. At ≤ 2 cm, detection is essentially saturated across all classes (Table 19), indicating that the preprocessing and classifier reliably capture the gesture signatures when coupling is strong. Beyond 3 cm, detection rates begin to drop and become class-dependent. In particular, *left* degrades fastest (reaching 0% at 5 cm), while *up* remains comparatively resilient (40%

at 5 cm). This spread is consistent with the directional nature of the gestures and small asymmetries in electrode coupling and hand trajectories.

Table 19: Gesture detection rate under varying hand–sensor distances.

Class	1 cm	2 cm	3 cm	4 cm	5 cm
left	100%	100%	60%	20%	0%
right	100%	100%	60%	40%	20%
up	100%	80%	100%	40%	40%
down	100%	100%	80%	40%	20%

Confidence vs Distance

Table 20 shows the mean confidence of the accepted detections. Confidence is near 1.0 at short distances and decreases progressively with distance, especially from 3 cm onward. Importantly, the reported confidence values are *conditional* on valid detections; the absence of a confidence entry (e.g., *left* at 5 cm) indicates that no valid detections occurred in that condition. The overall trend supports the interpretation that distance primarily reduces SNR, leading to both fewer accepted detections and lower confidence when detections still occur.

Table 20: Average confidence of valid detections vs distance.

Class	1 cm	2 cm	3 cm	4 cm	5 cm
left	1.00	0.97	0.92	0.92	
right	1.00	0.99	0.67	0.85	0.68
up	1.00	1.00	0.80	0.69	0.78
down	1.00	0.97	0.83	0.75	0.65

Rejections vs Non-detections

In addition to incorrect or missing classifications, the firmware can intentionally suppress low-confidence outputs via `PRED_EVT_IGNORED`. Across all distance trials, only ~ 5 such rejections were observed, meaning the majority of non-detections at larger distances were not explicit rejections but rather cases where the gesture did not produce a sufficiently separable signature for a correct class emission. In practice, this indicates that the dominant failure mode in the far-field regime is reduced gesture observability (low coupling), rather than overly aggressive confidence gating. This behavior is desirable from a user-experience standpoint: when signal conditions are poor, the system tends to remain quiet rather than producing frequent false triggers.

Takeaway

Overall, the field test confirms a clear operational envelope. Reliable performance is achieved at 1–2 cm, while performance degrades gradually beyond 3 cm with increasing class dependence. These results provide practical guidance for mechanical integration and user interaction distance, and motivate future improvements in electrode geometry and/or analog front-end sensitivity if a larger operating distance is required.

Chapter Summary

This chapter validated the complete capacitive gesture recognition system through a series of functional and performance-oriented tests spanning sensing, signal processing, classification, embedded execution, and field operation. The sensor front-end was shown to provide stable and sufficiently low-noise measurements, and frequency-domain analysis confirmed that gesture information is concentrated at low frequencies, supporting the chosen filtering strategy.

Timing measurements demonstrated that both raw acquisition and post-processing pipelines operate with stable and predictable cadence, ensuring consistent temporal alignment for gesture inference. Evaluation of the digital filtering pipeline showed effective noise suppression and baseline stabilization, with acceptable signal attenuation that does not impact classification performance.

The gesture classifier exhibited strong and consistent performance in offline evaluation and maintained reliable behavior when deployed on the embedded platform. On-device measurements confirmed that the model fits within the memory and latency constraints of the target microcontroller and supports real-time inference.

Field testing across varying hand–sensor distances established a clear operational range, with reliable near-field detection and graceful degradation at larger distances. Importantly, failure modes were dominated by missed detections rather than false activations, indicating robust and conservative system behavior. Overall, the results confirm that the implemented system meets its functional and real-time requirements and is suitable for practical deployment.

Chapter 6

Conclusion

This dissertation set out to explore the transformation of domestic partition walls from passive architectural elements into active, interactive interfaces, grounded in the principles of Human-Building Interaction (HBI) and Ambient Intelligence. Motivated by the increasing demand for unobtrusive, intuitive, and privacy-preserving interaction mechanisms in smart home environments, the work focused on the design, implementation, and validation of a modular gesture recognition system based on capacitive sensing and embedded machine learning.

6.1 Summary of Contributions

The primary objective of this work was the development of a gesture-based interaction module capable of enabling contactless control of domestic systems, with particular emphasis on integration within wall partitions. To achieve this, a complete system was conceived and realized, encompassing hardware design, firmware development, signal processing, machine learning integration, and system-level validation.

A key contribution of this dissertation is the design and implementation of a standalone Gesture Recognition Module (GRM) based on capacitive proximity sensing. The module was engineered to be compact, wall-mountable, and compatible with standard electrical switch form factors, supporting unobtrusive integration into domestic environments. The sensing surface, composed of custom-shaped electrodes and shield rings, demonstrated that capacitive sensing can be effectively embedded behind solid materials while maintaining adequate sensitivity for free-space gesture detection.

Another significant contribution lies in the signal processing and embedded machine learning pipeline developed for robust gesture recognition. Raw capacitive signals were shown to be susceptible to noise, drift, and environmental variability—an inherent limitation of proximity-based capacitive systems. To address this, a multi-stage digital processing pipeline was implemented, including baseline stabilization, filtering, decimation, and temporal windowing. On top of this pipeline, a lightweight neural network model

was trained, quantized, and deployed using TensorFlow Lite for Microcontrollers, enabling real-time inference directly on the ESP32 microcontroller.

The dissertation further demonstrated the practical applicability of the gesture module through its integration with an existing LED-based partition wall system. By decoupling the interaction mechanism from the actuation layer, the GRM was shown to function as a generic input device capable of controlling not only LED walls but, in principle, any smart home actuator. This architectural separation reinforces the modularity and extensibility of the proposed solution.

6.2 Evaluation of Objectives and Requirements

The objectives defined at the outset of this work were largely met. The system successfully enables non-contact, gesture-based interaction without relying on cameras or wearable devices, thereby addressing privacy concerns commonly associated with vision-based approaches. The use of capacitive sensing allows operation through opaque materials, preserving the visual and architectural integrity of the environment.

From a performance standpoint, the experimental results presented in Chapter 5 confirm that the proposed system meets the functional requirements established during the design phase. Sensor characterization demonstrated stable operation with manageable noise and drift levels after baseline compensation. Timing analysis confirmed deterministic sampling and inference behavior, with end-to-end latency compatible with real-time interaction.

Classifier evaluation showed that the embedded machine learning model achieved high recognition accuracy across the defined gesture set, both in offline testing and during on-device deployment. Field tests further demonstrated that although the system achieved acceptable performance at close range, it didn't demonstrate the full range of use expected.

Importantly, these results were achieved within the computational and memory constraints of an embedded microcontroller platform, confirming the feasibility of deploying learning-based gesture recognition at the edge without reliance on cloud processing.

6.3 Limitations

Despite the positive outcomes, several limitations must be acknowledged. Capacitive sensing remains inherently sensitive to environmental conditions, including humidity, nearby objects, and user grounding effects. While the implemented processing and learning pipeline mitigates many of these issues, absolute distance estimation and fine-grained gesture discrimination remain challenging.

Additionally, the gesture vocabulary evaluated in this work is intentionally limited, focusing on a small set of discrete gestures suitable for reliable recognition. Expanding this vocabulary would likely require additional electrodes, more complex models, or user-specific adaptation mechanisms.

Finally, the experimental validation was conducted in controlled and semi-controlled environments. Long-term deployment in real homes, with multiple users and varying conditions, may reveal additional challenges related to calibration, personalization, and robustness over time.

6.4 Future Work

Several directions for future work naturally emerge from this dissertation. From a hardware perspective, the exploration of alternative electrode geometries, materials, or multi-panel sensor arrangements could improve spatial resolution and robustness. Adaptive or self-calibrating sensing techniques could further reduce sensitivity to environmental drift. The main improvement would be to upscale the system to cover large surfaces and even overlap with actuator technologies, providing either bigger electrode area to increase range or increase in complexity of sensing motions with a more complex electrode grid.

Still on hardware improvements, a second version of the [GRM](#) could bring major improvement with the implementation of a PCB and the addition of hardware filters to reduce digital processing and minimize impact on sensitivity.

On the software side, future research could investigate adaptive or personalized machine learning models that evolve with user behavior, as well as continuous gesture recognition schemes that go beyond discrete classification. Multi-modal interaction, combining capacitive sensing with other privacy-preserving sensors, also represents a promising avenue.

At the system level, broader integration with existing smart home ecosystems, accessibility-oriented interfaces, or context-aware automation—would further validate the relevance of interactive partitions as a core element of future domestic environments.

6.5 Final Remarks

In conclusion, this dissertation demonstrates that interactive home partitions, enabled by capacitive gesture recognition and embedded machine learning, represent a viable and compelling direction for smart home interaction. By merging architectural elements with intelligent sensing and computation, the proposed approach advances the vision of ambient intelligence in which technology recedes into the background while enhancing everyday human experience.

The work contributes not only a functional prototype but also a design methodology and system architecture that can inform future developments at the intersection of electronics, human-computer interaction, and architectural design.

Bibliography

- [1] P. Remagnino and G. Foresti. "Ambient Intelligence: A New Multidisciplinary Paradigm." In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 35.1 (2005), pp. 1–6. doi: [10.1109/TSMCA.2004.838456](https://doi.org/10.1109/TSMCA.2004.838456).
- [2] M. Weiser. "Hot topics-ubiquitous computing." In: *Computer* 26.10 (1993), pp. 71–72. doi: [10.1109/2.237456](https://doi.org/10.1109/2.237456).
- [3] P. S. Samal. "Transforming Everyday Environments: the Power of Ambient Intelligence." In: (June 2024). Available at SSRN: <https://ssrn.com/abstract=4900298> or <http://dx.doi.org/10.2139/ssrn.4900298>.
- [4] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. "Context Aware Computing for The Internet of Things: A Survey." In: *IEEE Communications Surveys & Tutorials* 16.1 (2014), pp. 414–454. doi: [10.1109/SURV.2013.042313.00197](https://doi.org/10.1109/SURV.2013.042313.00197).
- [5] B. Becerik-Gerber, G. Lucas, A. Aryal, et al. "The field of human building interaction for convergent research and innovation for intelligent built environments." In: *Scientific Reports* 12 (2022), p. 22092. doi: [10.1038/s41598-022-25047-y](https://doi.org/10.1038/s41598-022-25047-y). url: <https://doi.org/10.1038/s41598-022-25047-y>.
- [6] H. S. Alavi, E. F. Churchill, M. Wiberg, D. Lalanne, P. Dalsgaard, A. Fatah gen Schieck, and Y. Rogers. "Introduction to Human-Building Interaction (HBI)." en. In: *ACM Transactions on Computer-Human Interaction* 26.2 (Mar. 2019), pp. 1–10. doi: [10.1145/3309714](https://doi.org/10.1145/3309714). url: <http://dx.doi.org/10.1145/3309714>.
- [7] S. E. Bibri. "Implicit and Natural HCI in Aml: Ambient and Multimodal User Interfaces, Intelligent Agents, Intelligent Behavior, and Mental and Physical Invisibility." In: *The Human Face of Ambient Intelligence: Cognitive, Emotional, Affective, Behavioral and Conversational Aspects*. Paris: Atlantis Press, 2015, pp. 259–318. isbn: 978-94-6239-130-7. doi: [10.2991/978-94-6239-130-7_6](https://doi.org/10.2991/978-94-6239-130-7_6). url: https://doi.org/10.2991/978-94-6239-130-7_6.

-
- [8] V. V. Kumari, Swarup, Bhuvaneshwari, Chandan, and N. Madhuri. "Gesture-Driven Smart Homes: A Survey of Control Systems and Flexible Sensors." In: *TIJER - International Research Journal* 11.5 (May 2024), a231–a233. issn: 2349-9249. url: <https://tijer.org/TIJER/papers/TIJER2405036.pdf>.
 - [9] A. Braun, T. Dutz, and F. Kamieth. "Capacitive sensor-based hand gesture recognition in ambient intelligence scenarios." In: *Proceedings of the 6th International Conference on Pervasive Technologies Related to Assistive Environments*. PETRA '13. Association for Computing Machinery, 2013. isbn: 9781450319737. doi: [10.1145/2504335.2504340](https://doi.org/10.1145/2504335.2504340). url: <https://doi.org/10.1145/2504335.2504340>.
 - [10] D. Sarma and M. K. Bhuyan. "Methods, Databases and Recent Advancement of Vision-Based Hand Gesture Recognition for HCI Systems: A Review." In: *SN Computer Science* 2.6 (Aug. 2021), p. 436. issn: 2661-8907. doi: [10.1007/s42979-021-00827-x](https://doi.org/10.1007/s42979-021-00827-x). url: <https://doi.org/10.1007/s42979-021-00827-x>.
 - [11] S. S. Rautaray and A. Agrawal. "Vision based hand gesture recognition for human computer interaction: a survey." In: *Artificial Intelligence Review* 43.1 (Jan. 2015), pp. 1–54. issn: 1573-7462. doi: [10.1007/s10462-012-9356-9](https://doi.org/10.1007/s10462-012-9356-9). url: <https://doi.org/10.1007/s10462-012-9356-9>.
 - [12] S. Ahmed, K. D. Kallu, S. Ahmed, and S. H. Cho. "Hand gestures recognition using radar sensors for human-computer-interaction: A review." en. In: *Remote Sens. (Basel)* 13.3 (Feb. 2021), p. 527.
 - [13] H. Lee, J. K. Mandivarapu, N. Ogbazghi, and Y. Li. "Real-time Interface Control with Motion Gesture Recognition based on Non-contact Capacitive Sensing." In: *CoRR* abs/2201.01755 (2022). arXiv: [2201.01755](https://arxiv.org/abs/2201.01755). url: <https://arxiv.org/abs/2201.01755>.
 - [14] T. Grosse-Puppenthal, C. Holz, G. Cohn, R. Wimmer, O. Bechtold, S. Hodges, M. Reynolds, and J. Smith. "Finding Common Ground: A Survey of Capacitive Sensing in Human-Computer Interaction." In: May 2017. doi: [10.1145/3025453.3025808](https://doi.org/10.1145/3025453.3025808).
 - [15] LibreTexts. *Dielectric Polarization*. Accessed: 2025-07-14. n.d. url: https://eng.libretexts.org/Workbench/Materials_Science_for_Electrical_Engineering/06%3A_Thermal_Optical/6.01%3A_Optical_Properties/6.1.01%3A_Dielectric_Polarization.
 - [16] Texas Instruments. *CapTIvate™ Technology Guide – Basics*. Accessed: 2025-07-14. n.d. url: https://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/CapTIvate_Design_Center/latest/exports/docs/users_guide/html/CapTIvate_Technology_Guide_html/markdown/ch_basics.html.
 - [17] H. Lee and Y. Kim. In: *SSRN Electronic Journal* (Jan. 2023). doi: [10.2139/ssrn.4357004](https://doi.org/10.2139/ssrn.4357004).

-
- [18] Texas Instruments. *Capacitive Sensing Basics*. Application Report SNOA92A. Revised May 2013, Accessed: 2025-07-14. Texas Instruments, May 2013. url: <https://www.ti.com/lit/an/snoa92a/snoa92a.pdf>.
- [19] V. Kumar, K. Preeti, V. Saini, A. Kaushik, and S. Sharma. "Interdigitated Electrodes (IDEs)-Supported Biosensing for Efficient Point-of-Care Applications." In: *ECS Sensors Plus* 3 (Oct. 2024). doi: [10.1149/2754-2726/ad8b59](https://doi.org/10.1149/2754-2726/ad8b59).
- [20] E.-C. Yang, Y. Chen, J.-Y. Wu, R. Chena, and C.-Y. Lo. "Enhancing the Detection Sensitivity in Capacitive Tactile Sensors With Optimized Electrode Shapes." In: *IEEE Sensors Journal* PP (June 2021), pp. 1–1. doi: [10.1109/JSEN.2021.3087253](https://doi.org/10.1109/JSEN.2021.3087253).
- [21] S. Bian and P. Lukowicz. "Capacitive Sensing Based On-board Hand Gesture Recognition with TinyML." In: *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers (UbiComp/ISWC '21 Adjunct)*. New York, NY, USA: ACM, 2021, pp. 4–5. doi: [10.1145/3460418.3479287](https://doi.org/10.1145/3460418.3479287). url: <https://doi.org/10.1145/3460418.3479287>.
- [22] A. Jaramillo-Yáñez, M. E. Benalcázar, and E. Mena-Maldonado. "Real-Time Hand Gesture Recognition Using Surface Electromyography and Machine Learning: A Systematic Literature Review." In: *Sensors* 20.9 (2020). issn: 1424-8220. doi: [10.3390/s20092467](https://doi.org/10.3390/s20092467). url: <https://www.mdpi.com/1424-8220/20/9/2467>.
- [23] T. Wang, Y. Zhao, and Q. Wang. "A flexible iontronic capacitive sensing array for hand gesture recognition using deep convolutional neural networks." en. In: *Soft Robot*. 10.3 (June 2023), pp. 443–453. doi: [DOI:10.1089/soro.2021.0209](https://doi.org/10.1089/soro.2021.0209).
- [24] F. Noble, M. Xu, and F. Alam. "Static Hand Gesture Recognition Using Capacitive Sensing and Machine Learning." In: *Sensors* 23.7 (2023). issn: 1424-8220. doi: [10.3390/s23073419](https://doi.org/10.3390/s23073419). url: <https://www.mdpi.com/1424-8220/23/7/3419>.
- [25] M. M. Rahman, A. Uzzaman, F. Khatun, M. Aktaruzzaman, and N. Siddique. "A comparative study of advanced technologies and methods in hand gesture analysis and recognition systems." In: *Expert Systems with Applications* 266 (2025), p. 125929. issn: 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2024.125929>. url: <https://www.sciencedirect.com/science/article/pii/S0957417424027969>.
- [26] Promwad. *Embedded ML: How to Run Machine Learning Models on Microcontrollers*. <https://promwad.com/news/embedded-ml-microcontrollers-model-deployment>. Apr. 2025.

-
- [27] B. Zhou, D. Geissler, M. Faulhaber, C. E. Gleiss, E. F. Zahn, L. S. S. Ray, D. Gamarra, V. F. Rey, S. Suh, S. Bian, G. Joost, and P. Lukowicz. “MoCaPose: Motion Capturing with Textile-integrated Capacitive Sensors in Loose-fitting Smart Garments.” In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7.1 (Mar. 2023). doi: [10.1145/3580883](https://doi.org/10.1145/3580883). url: <https://doi.org/10.1145/3580883>.
- [28] M. Wilhelm, D. Krakowczyk, and S. Albayrak. “PeriSense: Ring-based multi-finger gesture interaction utilizing capacitive proximity sensing.” en. In: *Sensors (Basel)* 20.14 (July 2020), p. 3990.
- [29] Google AI Edge. *LiteRT for Microcontrollers*. <https://ai.google.dev/edge/litert/microcontrollers/overview>. Last updated 2024-08-30 UTC. 2024.
- [30] Arm Limited. *CMSIS-NN: CMSIS Neural Network Software Library*. User manual, GitHub / online documentation. Arm. 2023. url: <https://arm-software.github.io/CMSIS-NN/latest/index.html>.
- [31] L. Lai, N. Suda, and V. Chandra. *CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs*. 2018. arXiv: [1801.06601](https://arxiv.org/abs/1801.06601) [cs.NE]. url: <https://arxiv.org/abs/1801.06601>.
- [32] *Edge Impulse — The Leading Edge AI Platform*. <https://edgeimpulse.com/>. 2025.
- [33] Texas Instruments. *FDC1004 4-Channel Capacitance-to-Digital Converter for Capacitive Sensing Applications*. Data Sheet SNOSC5C. Texas Instruments, 2014. url: <https://www.ti.com/lit/ds/snosc5c/snosc5c.pdf?ts=1765381417634>.
- [34] Texas Instruments. *Capacitive Proximity Sensing Using the FDC1004*. Application Report snoa928a. Texas Instruments, 2015. url: <https://www.ti.com/lit/an/snoa928a/snoa928a.pdf>.
- [35] Texas Instruments. *Capacitive Sensing: Ins and Outs of Active Shielding*. Application Report snoa926a. Texas Instruments, 2015. url: <https://www.ti.com/lit/an/snoa926a/snoa926a.pdf>.