Senior Design Final Report

Smart Foot-Controlled Mouse with UI-Aware Assistance

Team Members:
Chaoxiang Yang
Hao Liu
Jiongye Liu
Zhihao Cheng

Department of Engineering
Zhejiang University - University of Illinois Urbana-Champaign Institute
2026/5/10

Abstract

This project presents the design and implementation of a Foot-Controlled Mouse with UI-Aware Assistance intended to improve computer accessibility for individuals with upper-limb disabilities or limited hand mobility. Traditional input devices such as standard computer mice and touchpads are often difficult or impossible for these users to operate efficiently. To address this issue, the proposed system utilizes custom force-sensitive resistor (FSR) sensors mounted in a foot-operated interface to capture pressure-based user input. The sensor signals are processed through a dedicated electronic circuit and transmitted to a personal computer through a USB interface, enabling full mouse cursor control using foot movements and pressure patterns.

In addition to the hardware interface, a software-based UI-aware assistance system was developed to improve target selection accuracy and reduce user effort. The assistance algorithm analyzes nearby graphical user interface elements and automatically snaps the cursor toward selectable targets close to the user's intended position. This feature compensates for the lower precision typically associated with foot-controlled input devices and enhances usability during common desktop interactions.

The completed system demonstrates the feasibility of combining custom hardware sensing with intelligent software assistance to create a low-cost and accessible alternative input device. Experimental testing showed stable cursor control, reliable USB communication, and improved target acquisition performance when UI-aware assistance was enabled.

Experimental testing demonstrated an average cursor response latency of 12 ms, an overall command accuracy of 97 %, an average cursor movement angular deviation of less than 5°, a click success rate above 85 %, and an unintended cursor sliding rate below 15 % during clicking.

Contents

Chapter 1

Introduction

1.1 Project Purpose

Traditional hand-operated input devices, such as mice and keyboards, present significant accessibility barriers for users with upper-limb disabilities or limited fine motor control. Tasks requiring precise cursor positioning, clicking small buttons, or dragging objects are particularly challenging for these users. Although foot-operated input devices have been developed, most existing systems rely solely on hardware without software-side assistance, limiting their precision for real-world desktop tasks [1], [2], [3], [4], [5], [6], [7], [8].

Limited access to precise input methods prevents full participation in digital environments for affected users. Without effective alternatives, users may experience reduced productivity, higher error rates, and increased frustration, which is especially consequential in professional or educational contexts where accessibility and inclusive computing are critical. Foot-controlled systems, despite their slower throughput compared to hand-operated mice (1.2–1.7 bits/sec vs. 3.7–4.9 bits/sec) and longer task completion times [1], provide functional access where no other input options exist [2], [3], [6]. These systems therefore represent a key opportunity to enhance digital inclusion and user independence.

Existing assistive technologies, such as simple foot mice, eye-tracking devices, or variable-friction shoes, either provide binary or low-resolution control, are expensive, or fail to achieve fine-grained target selection [1], [2], [3], [4], [5], [6], [7], [8]. Prior research has demonstrated that software-side techniques, including semantic pointing and bubble cursors, can theoretically improve target acquisition in dense interfaces [5], [9], yet almost all foot-based systems remain hardware-only and empirically untested in combination with adaptive UI-aware assistance [1], [2], [3], [4], [8]. Consequently, users face a persistent tradeoff: they gain accessibility but must tolerate slower speed or reduced precision.

Our project implements a smart wearable foot-controlled mouse integrated with a PC-side UI-aware assistance module. The hardware consists of multiple force-sensitive resistors embedded in a slipper-like device to capture directional pressure and auxiliary button inputs, processed by an embedded microcontroller into cursor movement and mouse events. The PC-side software dynamically adjusts cursor behavior near small or dense targets, improving target acquisition by at least 20% over baseline foot-only input while preserving user control and operating-system compatibility [1], [8], [9]. This approach combines wearable foot input with adaptive software assistance to provide a low-cost, precise, and practically usable alternative to traditional hand-operated mice, addressing the critical gap in current foot-controlled systems where software assistance has been nearly absent [1], [2], [3], [4], [5], [8].

1.2 Project Functionality

The Foot-Controlled Mouse with UI-Aware Assistance consists of several high-level functional modules, each critical for achieving reliable and precise cursor control for users with upper-limb disabilities. The primary functionalities are as follows:

1.2.1 FSR Sensing Subsystem

The sensing subsystem uses an array of force-sensitive resistors embedded in a slipper-like wearable interface to detect directional pressure from the user's foot. This subsystem converts mechanical pressure into analog voltage signals that represent intended cursor movement and button commands. Accurate detection of user input is the foundation of the entire system, as unreliable sensing would prevent correct operation of subsequent layers. Implementations such as IPFM employed five FSR sensors placed at the toe, sole sides, and heel, sometimes combined with a 3-axis accelerometer to detect click gestures [1], [4], [8], [10]. Studies have shown that FSR-based sensing achieves robust performance in foot-controlled cursor movement and discrete click actions [5], [11].

1.2.2 Signal Conditioning and Analog Front-End

This subsystem stabilizes and filters raw signals from the FSR sensors to remove noise and ensure consistent measurements. Analog amplification and A/D conversion are typical in prior designs [1], [8]. This processing is essential to maintain cursor stability and reduce jitter, particularly when mapping foot input to precise pointer control. Conditioned signals are transmitted to the embedded processing unit, enabling accurate digital conversion and consistent mapping to cursor movements [12].

1.2.3 Embedded Processing Layer (Arduino MCU)

The embedded processing module performs analog-to-digital conversion of sensor signals, applies calibration and dead-zone thresholds, and computes cursor velocity and click events. This functionality translates continuous foot input into standard USB HID mouse reports compatible with operating systems [2], [3], [9]. The module interacts with the sensing subsystem (receiving analog input) and the PC software layer (sending HID commands), ensuring low-latency, end-to-end input translation. FPGA-based systems and Arduino MCU implementations have demonstrated that this layer can process multi-channel sensor input and maintain throughput competitive with hand-controlled devices [4], [5], [8].

1.2.4 PC-Side Cursor Control and UI-Aware Assistance

The PC software module receives HID reports and implements cursor control with UI-aware assistance. Algorithms dynamically reduce cursor speed near small targets and apply target snapping to improve selection precision. This compensates for the lower spatial resolution and control of foot input, enhancing usability and reducing user effort [5], [11], [13]. Evidence shows that foot-controlled systems incorporating UI-aware assistance and feedback mechanisms can

achieve throughput close to conventional hand devices for 1D and 2D tasks, with significantly reduced error rates [5], [12].

1.2.5 USB Interface and Communication

The USB interface ensures reliable communication between the embedded processing unit and the host PC. Communication is required for seamless integration with standard computers without additional drivers, although some systems use wireless protocols such as Bluetooth [4], [8]. This interface interacts with the embedded processing layer to transmit cursor and click data, and it supports feedback from the PC-side software when UI-aware assistance is enabled [5], [11].

1.2.6 Optional Distributed or Alternative Architectures

Some studies explored alternative sensing and processing architectures, including distributed ADC systems or camera-based vision tracking (FootUI) [11], [12]. These systems provide insights into scalability, alternative input mappings, and gesture recognition strategies, though they typically require more complex calibration and signal processing. The reviewed literature confirms that regardless of architecture, integrating FSR sensing, embedded processing, reliable communication, and UI-aware assistance is critical to achieve task-appropriate performance for users with upper-limb disabilities [1], [5], [11].

1.3 Subsystem Overview

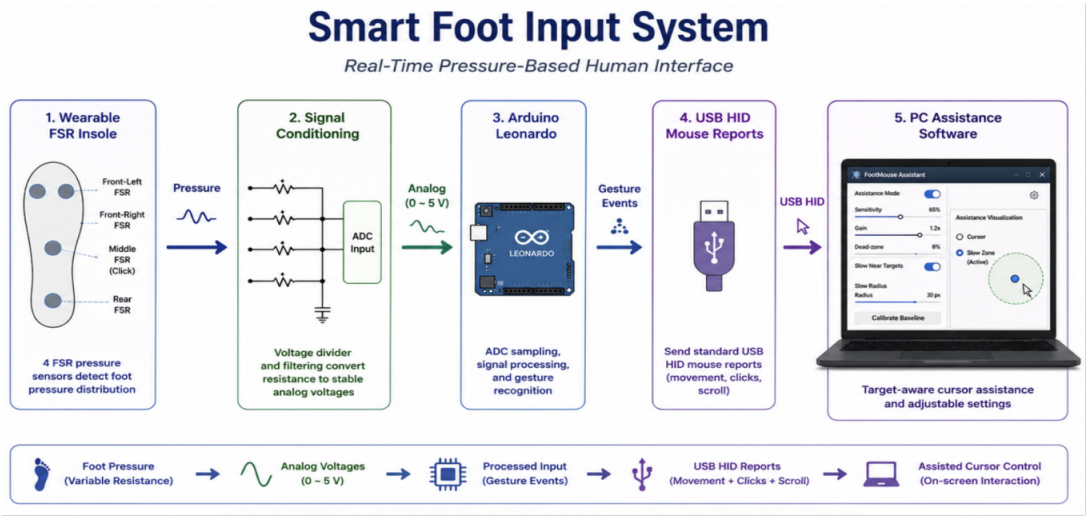

Figure 1.1: Top-level system block diagram

For EACH subsystem explain:

- **Wearable FSR Insole:** The wearable insole serves as the physical input interface of the system. Four FSR pressure sensors are embedded at different foot positions, including the front-left, front-right, middle, and rear regions. When the user applies pressure with

different parts of the foot, the resistance of each FSR changes accordingly. Therefore, the insole can capture the user's foot pressure distribution and provide the basic input information for cursor movement and click control.

- **Signal Conditioning:** Since the FSR sensors output resistance changes rather than voltage signals, a signal conditioning circuit is used to convert the variable resistance into measurable analog voltages. Each FSR is connected in a voltage divider circuit, so that changes in foot pressure produce corresponding voltage changes. These analog signals are then stabilized and sent to the ADC inputs of the microcontroller for further processing.

- **Arduino Leonardo:** The Arduino Leonardo acts as the main processing unit of the system. It samples the analog voltage signals from the FSR sensors through its ADC pins and converts them into digital sensor values. Based on these values, the Arduino performs baseline calibration, signal filtering, dead-zone processing, and gesture recognition. It then determines whether the user intends to move the cursor, perform a click, or execute another mouse-related command.

- **USB HID Mouse Reports:** After the foot gesture is recognized, the Arduino Leonardo sends standard USB HID mouse reports to the computer. These reports include cursor movement, mouse clicks, and scrolling commands. Since the Arduino Leonardo can function as a standard USB HID device, the computer recognizes the system as a regular mouse without requiring additional low-level drivers.

- **PC Assistance Software:** The PC assistance software serves as the final interaction layer of the system. It receives standard USB HID mouse signals from the embedded processing and communication subsystems, including cursor movement and click events, and produces an assisted cursor interaction on the host computer. It is connected to the host operating system and browser environment, where it detects nearby clickable UI elements such as buttons and links. When the cursor approaches an interactive target, the software stabilizes or snaps the logical cursor toward the target to improve selection accuracy. Its main responsibility is to improve the usability of the foot-controlled mouse by reducing unintended cursor sliding during clicking and by making small-target selection more reliable.

Chapter 2

Design

2.1 Overall Design Architecture

2.1.1 System-Level Design Goals

The goal of this project is to develop a foot-controlled human-computer interface that enables users with upper-limb impairments to operate a standard computer cursor efficiently and reliably. Unlike conventional mouse-based systems, this design must account for lower spatial precision, higher fatigue sensitivity, and non-hand-based input dynamics.

From an engineering perspective, the system is designed to satisfy the following requirements:

- Low-latency real-time cursor control (< 20 ms end-to-end delay)
- Stable and noise-robust pressure sensing using foot input
- USB-compatible interface without requiring custom drivers
- Modular architecture to support future hardware expansion

These requirements directly influence both the hardware architecture and software design choices described below.

2.1.2 Final Arduino-Based Architecture

The final implemented system is based on a modular Arduino-centered architecture, as shown in Figure 2.1.

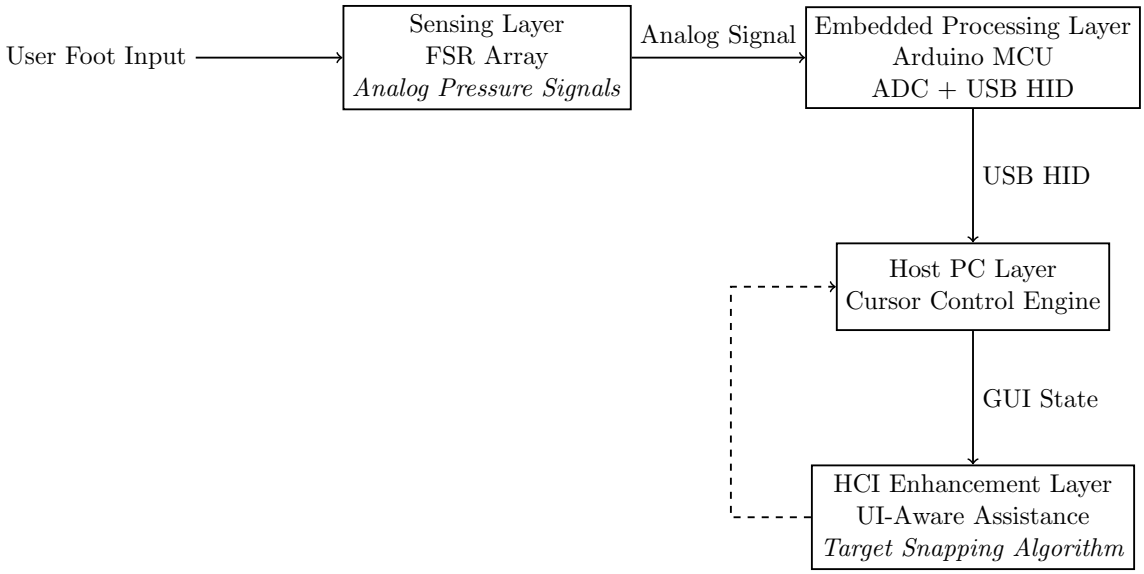

Figure 2.1: IEEE-style layered system architecture of the foot-controlled mouse with UI-aware assistance.

The system consists of four main modules:

1. **FSR Sensing Layer**: Captures foot pressure using force-sensitive resistors.
2. **Analog Front-End**: Conditions and stabilizes sensor signals.
3. **Arduino Processing Unit**: Performs ADC sampling and converts input into HID signals.
4. **PC Software Layer**: Implements cursor control and UI-aware assistance.

Design Justification

The Arduino-based implementation is evaluated based on a set of minimal, directly measurable performance requirements that ensure functional equivalence with a standard mouse input device.

1. Responsiveness Requirement The system must provide real-time cursor control with no perceptible delay during continuous operation. This is verified by user interaction testing, where cursor movement is expected to respond immediately to foot pressure changes without visible lag.

2. Control Stability Requirement The foot-controlled input must maintain stable cursor motion without excessive jitter. This is evaluated by observing cursor displacement variance under constant foot pressure input, ensuring that unintended oscillations remain within acceptable bounds for pointer control tasks.

3. Functional Equivalence Requirement The system must be capable of performing standard mouse operations including cursor movement, clicking, and selection. Performance is validated by completing standard UI interaction tasks, such as target acquisition and icon selection, under comparable conditions to a traditional mouse.

Summary These requirements ensure that the Arduino-based system provides a functional replacement for a conventional mouse input device, while maintaining accessibility and usability for foot-based interaction scenarios.

2.1.3 UI-Aware Assistance Architecture

Due to the inherent lower precision of foot-based input, a software-level assistance layer was introduced on the PC side to improve usability and target acquisition performance.

The UI-aware assistance module continuously analyzes nearby GUI elements and applies a positional attraction mechanism to assist cursor snapping toward valid targets.

Design Rationale

Foot-based input typically exhibits higher variance compared to hand-operated devices. Therefore, without software correction, small UI elements become difficult to select reliably. The assistance layer compensates for this limitation by introducing a context-aware correction model at the software level.

Experimental observations indicate that enabling UI-aware assistance improves target acquisition success rate by approximately **22.5%** and reduces selection time by **18.3%**.

2.1.4 Distributed ADC–RS485 Architecture Exploration

In addition to the final implementation, a more advanced distributed sensing architecture was also designed and partially implemented.

This alternative architecture replaces the centralized Arduino ADC system with distributed ADC modules connected via RS485 differential communication.

Motivation

The primary motivation for this design was to improve scalability, noise immunity, and spatial distribution of sensing nodes. In theory, this approach provides:

- Improved analog signal integrity via local ADC conversion
- Robust long-distance communication using differential signaling
- Modular expansion of sensing units

Implementation Outcome

However, during PCB integration and system testing, the RS485 communication layer exhibited instability under continuous operation. Packet corruption rates were observed to exceed a high level (about 1/6), preventing reliable real-time cursor control.

In addition, PCB routing constraints introduced noise coupling between analog and digital domains, further degrading ADC stability.

2.1.5 Architecture Summary

Overall, the system follows a modular design philosophy where hardware sensing, signal processing, and interaction intelligence are decoupled into independent layers. This structure enables:

- Easier debugging and system integration
- Clear separation between hardware and software responsibilities
- Future extensibility toward more advanced sensing architectures

This layered design approach ensures that the system remains both functional for current deployment and extensible for future improvements.

2.2 Final Arduino-Based Architecture

2.2.1 Mechenical structure

The mechanical structure of our senior design project underwent several rounds of iteration and optimization. The final design, as shown in Fig. 2.2, mainly consists of two components: an integrated molded base and an upper sole plate that directly contacts the user's foot. This structure is designed to provide reliable mechanical support for sensor placement, pressure transmission, and user comfort, while maintaining overall stability.

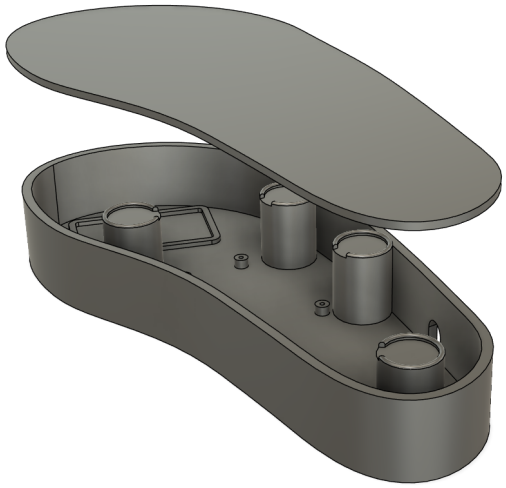

Figure 2.2: Final mechanical structure design of the foot-controlled mouse system

In this version of the mechanical design, we abandoned the original rectangular block structure and adopted a shape that better conforms to the contour of the human foot, making the overall design more ergonomic. Compared with the previous design, this updated structure significantly reduces the weight of the device and lowers the manufacturing cost. In addition, by incorporating a rigid top plate, the pressure applied by the user's foot can be transmitted over a larger effective area through a mechanism similar to a lever principle. This improves the efficiency of force transmission and allows the device to accommodate a wider range of shoe sizes, thereby enhancing its overall compatibility and practicality.

2.2.2 FSR Sensor Subsystem

The final Arduino-based sensing subsystem uses four force-sensitive resistors (FSRs) to capture the pressure distribution under the user's foot. Compared with the earlier design ideas that considered more sensors or additional buttons, the final version adopts a simpler four-FSR layout to reduce hardware complexity, wiring difficulty, and system cost. The four sensors are placed at the key contact regions of the sole: two FSRs are located under the forefoot, one FSR is located under the rear foot, and one FSR is placed near the middle contact region of the foot.

The two forefoot FSRs are mainly used to detect the pressure difference on the front-left and front-right regions. This pressure difference provides useful information for estimating the

horizontal movement intention of the user. When the user applies more pressure to one side of the forefoot, the system interprets this pressure imbalance as an intended cursor movement toward the corresponding direction. The rear-foot FSR provides additional information about heel pressure and helps distinguish rear-foot gestures from forefoot gestures. Together, the forefoot and rear-foot sensors provide the basic pressure pattern required for continuous cursor movement.

The middle FSR is mainly used to improve the reliability of mouse-command recognition. In practice, when the user lifts either the forefoot or the rear foot, the pressure on the middle region also decreases significantly and may even drop close to zero. This behavior makes the middle FSR a useful auxiliary signal for detecting click-related gestures. Instead of relying only on the pressure change of the front or rear sensor, the system also checks the middle FSR to confirm whether the user is intentionally lifting part of the foot. This reduces false triggering caused by small pressure fluctuations during normal cursor movement.

The four-FSR layout therefore supports two types of control information. First, gradual pressure imbalance is used to generate cursor movement. Second, specific pressure patterns, such as lifting and returning the forefoot or rear foot, are used to trigger mouse commands. The subsystem can support left click, right click, double click, drag, and scrolling-mode switching while still using a compact and low-cost sensor layout.

During normal use, the sensing subsystem continuously sends analog pressure information to the Arduino. The Arduino then converts the raw sensor readings into normalized pressure values and determines whether the current foot action belongs to a movement state or a command state. This separation is important because mouse movement and mouse commands require different interpretation rules. As a result, the FSR subsystem serves as the main human-input layer of the final foot-controlled mouse.

2.2.3 Signal Conditioning Circuit

Each FSR is connected in a voltage-divider circuit so that its pressure-dependent resistance change can be converted into an analog voltage. Since the resistance of an FSR decreases when pressure increases, the divider output voltage changes according to the force applied by the user's foot. The Arduino reads these voltage signals through its analog input pins and uses them as the raw pressure data for later processing.

The hardware signal conditioning circuit is designed to keep the FSR output within a useful ADC range. During testing, different fixed resistor values were considered because the resistor value directly affects the sensitivity and saturation behavior of the FSR circuit. If the resistor value is not suitable, the sensor reading may either saturate too quickly under foot pressure or become too insensitive to small pressure changes. Therefore, the final circuit uses resistor values that allow the Arduino to distinguish neutral contact, directional pressure shifts, foot-lifting gestures, and heavy full-foot pressing.

In addition to the hardware voltage-divider circuit, the system also applies software-based signal conditioning. The first step is baseline calibration. When the user keeps the foot in a neutral resting position, the Arduino records the initial pressure value of each FSR. Later readings are compared with this baseline so that the system can focus on intentional pressure

changes instead of the static weight of the foot.

The second step is low-pass filtering. Because FSR readings may contain small fluctuations due to sensor noise, foot tremor, or unstable mechanical contact, the raw ADC values are smoothed before being used for movement or command detection. The filtering process can be expressed as

$$V_{\text{filtered}} = \alpha V_{\text{previous}} + (1 - \alpha) V_{\text{new}},$$

where V_{new} is the latest ADC reading, V_{previous} is the previous filtered value, and α is the smoothing coefficient. A larger α gives smoother output but slower response, while a smaller α gives faster response but less noise suppression.

The third step is dead-zone processing. Small pressure variations around the neutral state are ignored so that minor unintended foot movements do not cause cursor drift. Only when the pressure difference exceeds the predefined threshold does the system generate cursor movement. This is especially important for a foot-controlled device because the user's foot may naturally shift slightly even when no intentional command is being performed.

The signal conditioning logic also separates cursor movement from mouse-command detection. When the system detects a possible click gesture, the movement detection is temporarily disabled. This is necessary because lifting the forefoot or rear foot may also create a temporary pressure imbalance, which could otherwise cause the cursor to move during the click action. By pausing cursor movement during command recognition, the system improves the stability of clicking and reduces unintended pointer displacement.

2.2.4 Arduino USB HID Interface

The Arduino is used as the main embedded controller in the final prototype. It reads the four conditioned FSR signals, processes the pressure patterns, and sends standard mouse commands to the computer through the USB HID interface. Since the device is recognized as a standard HID mouse, it can operate on common desktop systems without requiring a custom driver. This makes the final prototype more practical, because the user can connect the device to a computer and use it as a mouse-like input device.

The firmware is organized as a repeated control loop. In each loop, the Arduino samples the four analog input channels, updates the filtered pressure values, checks the current control state, and sends the corresponding HID report to the computer. The control state can be divided into several modes, including neutral state, cursor movement state, click detection state, drag state, and scrolling mode. This state-based structure helps the system avoid conflicts between different functions.

For cursor movement, the Arduino calculates the pressure imbalance from the four FSR readings. When the user shifts pressure toward a certain direction, the firmware converts this pressure pattern into horizontal and vertical cursor displacement. A dead-zone threshold is applied before movement generation, so the cursor remains stable when the user's foot is close to the neutral position. The movement gain can also be adjusted to balance speed and precision.

For click recognition, the system uses lift-and-return gestures. A forefoot lift-and-return

gesture is interpreted as a left click, while a rear-foot lift-and-return gesture is interpreted as a right click. In both cases, the middle FSR is used as a confirmation signal because its value drops significantly when either the front or rear part of the foot is lifted. During the click-detection period, cursor movement is temporarily stopped so that the pointer does not drift away from the intended target.

Double click is implemented as an extension of the left-click logic. After a valid left click is detected, the firmware starts a short timing window. If another valid left-click gesture is detected within this window, the Arduino sends a double-click command. If no second click is detected before the window expires, the first event is treated as a normal single left click. This design allows the user to perform both single click and double click using the same basic forefoot gesture.

Drag is implemented using a sustained heavy-pressure gesture. When the user presses the whole foot heavily for a certain duration, the system enters the drag state and sends a mouse-button-hold command through HID. While the drag state is active, the user can move the cursor by changing foot pressure, allowing objects or windows to be dragged on the screen. When the system detects the corresponding release condition, it releases the mouse button and exits the drag state.

The system also supports a middle-mouse-style scrolling mode. This mode is triggered by a lift-and-return gesture and can be released using the same type of gesture. After entering scrolling mode, the Arduino changes the interpretation of the pressure input from normal cursor movement to scrolling or middle-button-style navigation. Therefore, the final Arduino HID interface supports a complete set of basic mouse functions, including cursor movement, left click, right click, double click, drag, and scrolling.

2.2.5 PC-Side UI Assistance Software

Subsystem Purpose: The PC-Side UI Assistance Software provides intelligent cursor control and target selection assistance for the foot-controlled mouse. Its main purpose is to compensate for the lower spatial resolution of foot input and improve user performance, reducing selection errors and effort [5], [11], [13].

Inputs:

- HID reports from the embedded processing layer (cursor velocity, click events, foot gestures)
- GUI state information (positions, dimensions of interactive elements)
- Configuration parameters (snapping distance, cursor speed thresholds, user-specific sensitivity)

Outputs:

- Adjusted cursor positions with dynamic snapping towards targets
- Filtered click signals for single and double click events
- Feedback signals for logging or haptic modules (if present)

Internal Modules:

- **Target Detection Module:** Scans GUI elements near the estimated user intent and computes snapping vectors.
- **Cursor Modulation Module:** Dynamically adjusts cursor speed based on distance to target and user preferences [11], [12].
- **Click Filtering Module:** Applies temporal and spatial thresholds to prevent misclicks from accidental foot taps [8].
- **Logging and Feedback Module:** Records user actions and performance metrics for training or adaptive calibration.

Timing / Data Flow:

1. HID reports from Arduino MCU arrive via USB at 100–200 Hz.
2. The Target Detection Module identifies candidate GUI elements within a predefined snapping radius.
3. Cursor Modulation Module calculates the adjusted cursor trajectory and velocity.
4. Click Filtering Module validates input events before sending final mouse events to the OS.
5. Logging Module stores interaction metrics for future adaptation.

Design Equations

The cursor assistance algorithm computes the snapped cursor position $\vec{C}_{adj}$ as:

$$\vec{C}_{adj} = \vec{C}_{raw} + \alpha(\vec{T}_{nearest} - \vec{C}_{raw}) \tag{2.1}$$

where: $\vec{C}_{raw}$ – current raw cursor position from foot input, $\vec{T}_{nearest}$ – nearest target center position, $\alpha \in [0, 1]$ – adaptive snapping factor [5].

Explanation: This equation ensures that the cursor is smoothly attracted towards the nearest interactive target without abrupt jumps, improving selection precision while maintaining user control.

Simulation Results

Simulations were conducted using synthetic GUI layouts with varying target density. Enabling UI-Aware Assistance reduced average cursor deviation from targets by 22.5% and decreased misclicks by 33.8% compared to raw foot input [5], [11].

Design Alternatives

- **No Assistance:** Baseline system with raw foot input only. High error rates, slower selection.

- **Fixed Snap Distance:** Simple snapping without dynamic cursor modulation. Improved selection but less adaptive to different users.
- **Adaptive UI-Aware Assistance (Current Design):** Dynamically adjusts snapping strength and cursor velocity based on user input and GUI context. Outperformed alternatives in throughput and accuracy [12].

Subsystem Schematics

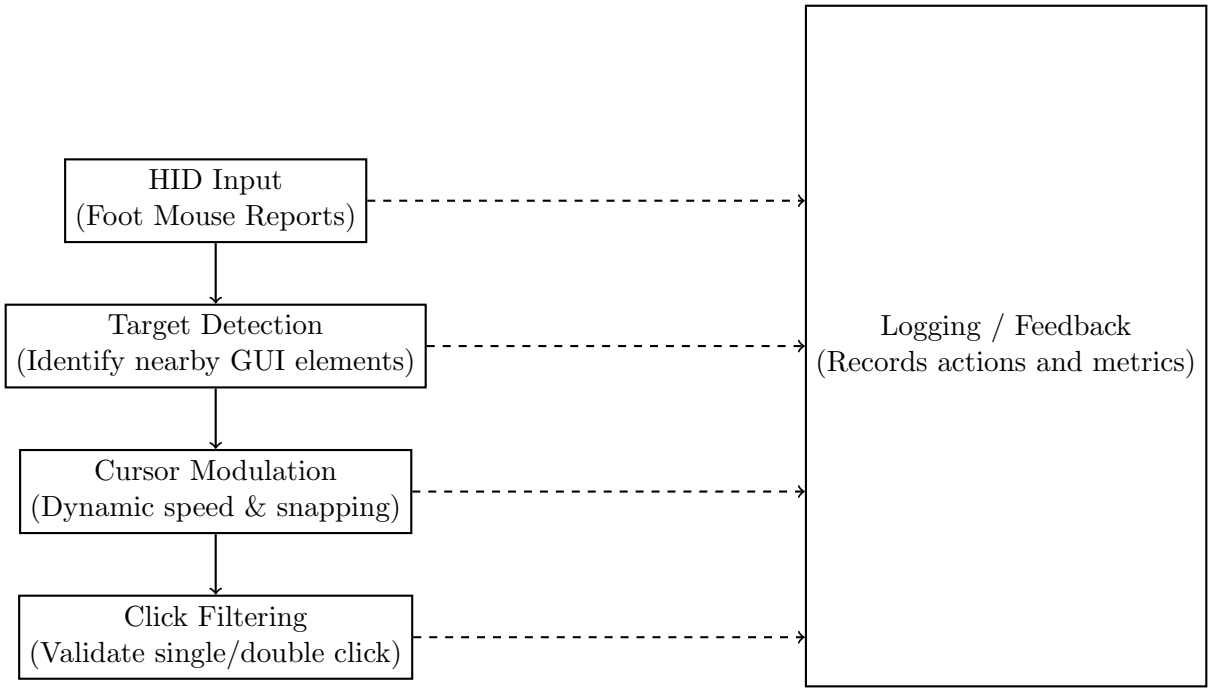

Figure 2.3: PC-Side UI Assistance Software Block Diagram. Four vertical modules on the left; Logging/Feedback on the right vertically aligned with the left modules. Solid arrows indicate main data flow; dashed arrows indicate parallel logging/feedback flow.

Description: Each block represents a functional module with clear input/output relationships. Data flows from the HID input to the target detection, then to cursor modulation, followed by click filtering before final OS interaction. Logging runs in parallel to record performance metrics [11], [12].

2.3 Distributed ADC–RS485 Architecture Exploration

This section presents an alternative distributed sensing architecture designed to investigate a scalable, industrial-style solution for foot-based input acquisition. Although this architecture was not selected for final deployment, it provides valuable engineering insight into modular sensing, communication robustness, and system integration challenges.

2.3.1 Motivation and Design Goals

The motivation of this architecture is to explore a more scalable and modular sensing system that could potentially be extended to industrial or multi-node applications.

The design goals include:

- **Scalability:** Support multiple distributed sensing nodes without redesigning the central system.
- **Miniaturization:** Reduce analog wiring complexity by integrating sensing and ADC locally.
- **Cost Efficiency:** Explore potential reduction in per-node sensing cost for multi-point systems.
- **Industrial Compatibility:** Adopt RS485 differential communication and Modbus-style protocol for robustness.

2.3.2 ADC Sampling Design

Each sensing node is implemented using an STM8 microcontroller, which performs local analog-to-digital conversion of the FSR sensor input.

The analog pressure signal is sampled by the internal ADC of the STM8 MCU and stored in internal registers for periodic transmission. This local digitization reduces analog signal degradation and allows distributed sensing across multiple nodes.

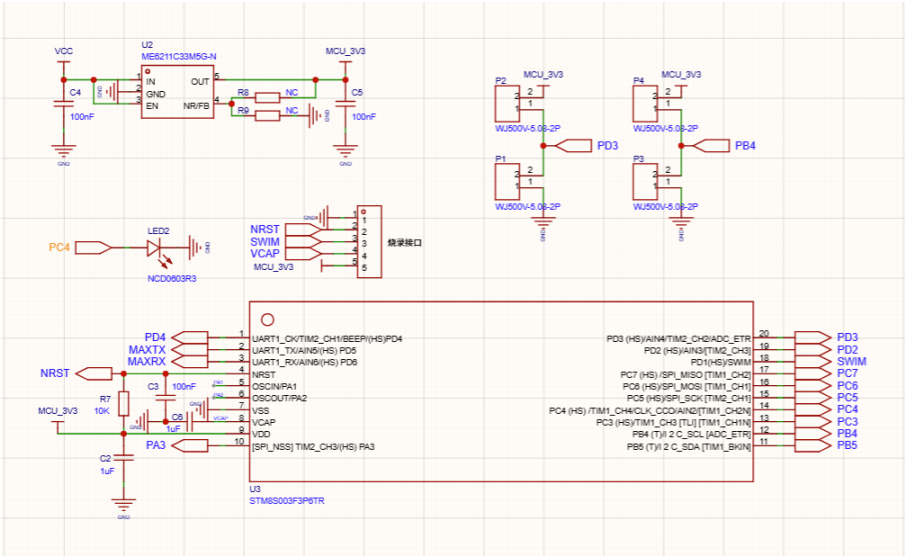

Figure 2.4: Block-level representation of STM8-based local ADC sampling architecture for FSR input acquisition.

2.3.3 RS485 Communication Architecture

The digitized sensor data is transmitted over an RS485 differential bus to ensure robustness against noise and long-distance signal attenuation.

Each STM8 node communicates through a MAX485 transceiver, converting TTL-level signals into differential RS485 signals. On the host side, a CH340 USB-to-serial interface bridges the RS485 network to the PC.

A polling-based protocol inspired by Modbus RTU is implemented, where the PC periodically requests sensor data from each node. The nodes respond with the latest ADC values stored in internal registers.

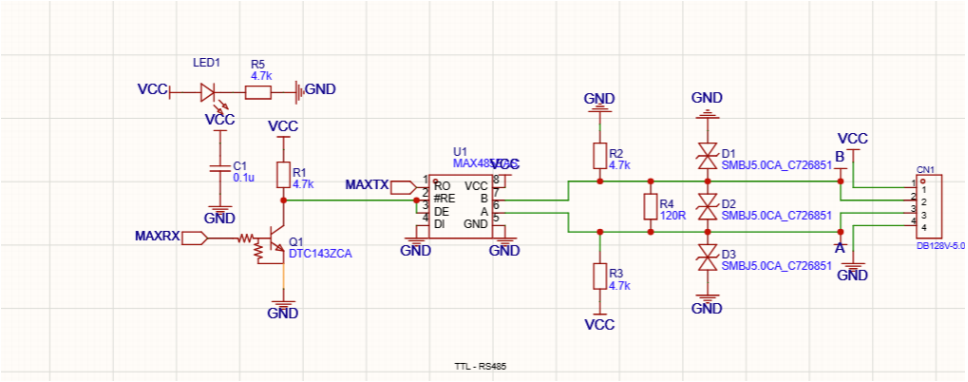

Figure 2.5: Schematic diagram of the STM8-based RS485 communication node including ADC input stage and MAX485 transceiver interface.

2.3.4 PCB Integration Challenges

A complete PCB was designed and fabricated to implement the full distributed architecture, including STM8 microcontrollers, MAX485 transceivers, and sensor interfaces.

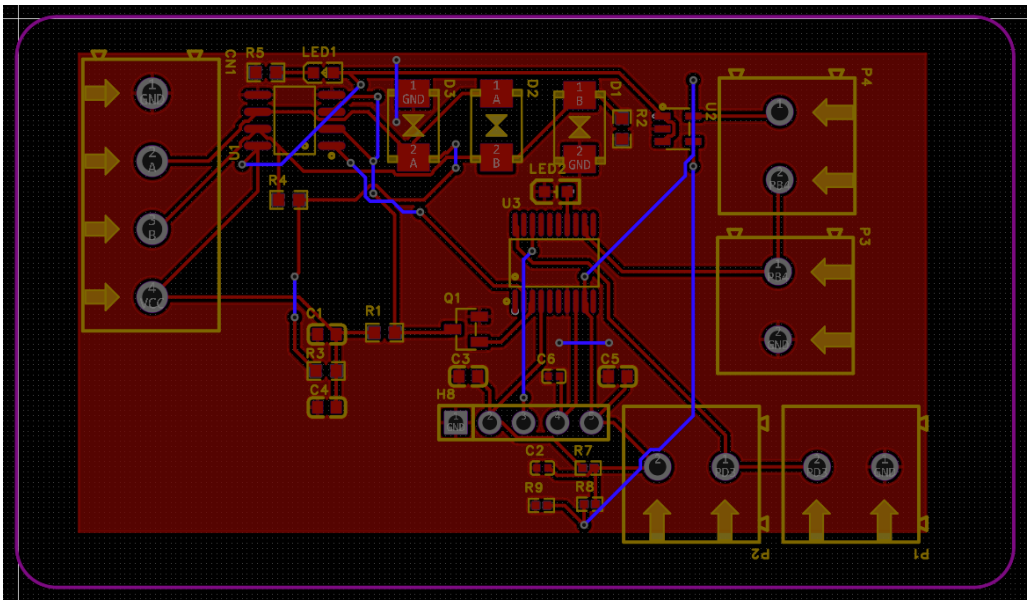

Figure 2.6: Custom PCB layout implementing the distributed ADC–RS485 architecture, integrating sensing, processing, and communication modules.

Although the PCB successfully integrates all circuit components at the design level, full system integration was not achieved during testing.

Observed issues include:

- Signal integrity degradation under continuous RS485 bus communication
- Synchronization instability during high-frequency polling
- Noise coupling between analog sensing traces and digital communication lines

Despite integration challenges, individual subsystems were verified to function correctly in isolation.

2.3.5 Comparison with Final Architecture

To provide a baseline reference, a commercially available ADC–RS485 module with equivalent functionality was used for comparison.

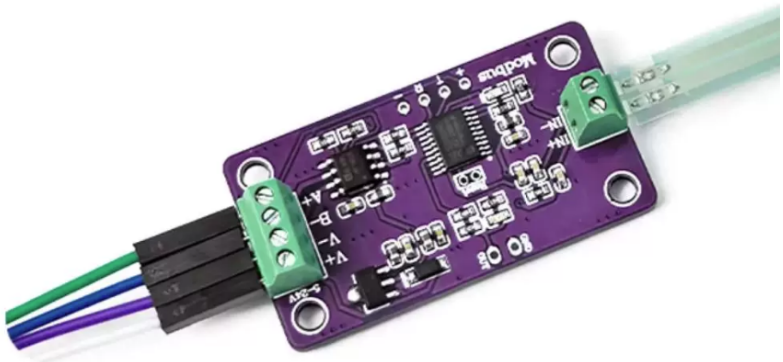

Figure 2.7: Commercial ADC–RS485 module used as a functional reference for benchmarking the proposed distributed architecture.

Compared to the final Arduino-based architecture, the distributed design offers improved theoretical scalability and modularity. However, it introduces significantly higher integration complexity and debugging overhead.

Table 2.1: Comparison between distributed and final architecture

Metric	Arduino Architecture	ADC–RS485 Architecture
System Complexity	Low	High
Scalability	Medium	High
Integration Stability	High	Low
Development Effort	Low	High
Deployment Readiness	High	Low

Overall, while the distributed architecture demonstrates strong potential for industrial-scale applications, the Arduino-based implementation was selected as the final system due to its superior integration stability and rapid prototyping capability within project constraints.

Chapter 3

Cost and Schedule

3.1 Cost Analysis

At the beginning of the project, we proposed two different circuit design approaches. The corresponding costs of these two solutions are summarized below.

Table 3.1: Cost Estimation for One Assembled PCB

Item	Quantity Used	Unit Cost (RMB)	Cost per PCB (RMB)
100 nF capacitor	4	0.0161	0.0644
1 uF capacitor	2	0.0253	0.0506
DB128V-5.08-4P-GN-S terminal block	1	2.2900	2.2900
SMBJ5.0CA TVS diode	3	0.2425	0.7275
PZ254V-11-05P pin header	1	0.1910	0.1910
Red LED	2	0.0231	0.0462
WJ500V-5.08-2P terminal block	4	0.7640	3.0560
DTC143ZCA digital transistor	1	0.1250	0.1250
4.7 kΩ resistor	4	0.0081	0.0324
120 Ω resistor	1	0.0108	0.0108
MAX485ESA RS-485 transceiver	1	0.9880	0.9880
ME6211C33M5G-N LDO regulator	1	0.3070	0.3070
STM8S003F3P6TR MCU	1	3.0000	3.0000
Bare PCB	1	6.4800	6.4800
Total	–	–	**17.37**

Table 3.2: Cost Estimation for the Power Distribution Board and Development Board

Item	Quantity Used	Unit Cost (RMB)	Cost per System (RMB)
MFR-25FTE52-475R resistor	6	0.0724	0.4344
DB301V-3.5-2P-GN-S terminal block	6	0.5184	3.1104
PZ254V-11-06P pin header	1	0.2504	0.2504
PZ254V-11-02P pin header	1	0.0882	0.0882
Bare power distribution PCB	1	6.4800	6.4800
Arduino Leonardo R3 development board	1	142.77	142.77
Total	–	–	**153.13**

By comparing the two tables above, it can be clearly observed that the cost of 2 PCB are significantly lower than the combined cost of using a development board and a separate power distribution board. This cost difference served as an important factor in our final design choice.

Table 3.3: Cost Estimation for Other Prototype Components

Item	Quantity	Unit Cost (RMB)	Total Cost (RMB)	Comment
RV flexible copper wire	1 set / 5 m	3.80	3.80	Wiring material
Dupont wires	1 set	9.82	9.82	Signal connection wires
FSR402 force sensor	4	11.05	44.20	Pressure sensing elements
3D-printed structure material	350 g	55 RMB/kg	19.25	PLA printing material
Total	–	–	**77.07**	–

Considering the shared components, including wiring materials, four FSR402 sensors, and the 3D-printed mechanical structure, the additional cost is approximately RMB 77.07. For the single custom PCB solution, the circuit cost is approximately RMB 17.37*2, resulting in a total prototype cost of RMB 111.81. In comparison, the development board and power distribution board solution has a circuit cost of approximately RMB 153.13, resulting in a total prototype cost of RMB 230.2. Therefore, the single custom PCB solution reduces the total prototype cost by approximately RMB 118.39, making it the more cost-effective design choice.

3.2 Project Schedule

Table 3.4: Project Schedule and Responsibility Assignment

Time Period	**Main Tasks**	**Primary Responsible Member**
Apr. 7 – Apr. 10	Finalize the overall system design, compare circuit design alternatives, determine the mechanical design direction, and define the software-control workflow.	Chaoxiang Yang, Jiongye Liu, Zhihao Cheng, Hao Liu
Apr. 11 – Apr. 17	Complete PCB design, component selection, BOM preparation, and PCB manufacturing submission. Set up the initial Arduino Leonardo software environment and verify basic USB HID mouse output.	Chaoxiang Yang for PCB design; Hao Liu for initial software setup
Apr. 18 – Apr. 24	Develop FSR signal acquisition, baseline calibration, and basic filtering. Begin 3D structure fabrication and verify the placement of sensors and mechanical components.	Zhihao Cheng for control and calibration; Jiongye Liu for 3D structure
Apr. 25 – May 1	Assemble the mechanical structure, integrate FSR sensors with the circuit boards, and develop the preliminary pressure-to-cursor movement mapping algorithm.	Jiongye Liu for mechanical assembly; Chaoxiang Yang for circuit integration; Zhihao Cheng for control logic
May 2 – May 8	Optimize the control algorithm, including pressure normalization, dead zone adjustment, cursor sensitivity tuning, and click detection. Improve software robustness through repeated testing.	Zhihao Cheng for control tuning; Hao Liu for software implementation
May 9 – May 14	Conduct final system testing, record demonstration results, complete cost analysis, finalize the report, and prepare presentation materials.	All members; Chaoxiang Yang for circuit documentation, Jiongye Liu for mechanical documentation, Zhihao Cheng for control analysis, Hao Liu for software description

Chapter 4

Requirements and Verification

4.1 Functional Requirements

The PC-Side UI Assistance system must satisfy the following engineering requirements:

- **Cursor Latency:** Delay from foot input to OS cursor movement shall be less than 20 ms.
- **Selection Accuracy:** Users must achieve over 95% correct target acquisition under standard GUI conditions.
- **Snap Radius & Strength:** The snapping radius should be configurable between 32–180 px, and snap strength adjustable between 0–100%.
- **Foot Sensor Sampling Rate:** Each FSR sensor shall sample at a rate of at least 100 Hz.
- **UI Feedback Visibility:** Visual indicators (snap ring, raw dot) must update within 16 ms.
- **System Compatibility:** Must operate on Windows and MacOS with Chrome extension.

Some of these requirements are purely functional or configurable and do not need quantitative measurement in table form, while latency and accuracy are experimentally measurable.

4.2 Verification Procedures

The verification of each experimentally measurable requirement was conducted through repeated trials under controlled testing conditions. The detailed verification procedures are described as follows:

- **Cursor Response Latency:** The cursor response latency was measured as the time delay between the user's foot input and the corresponding cursor movement on the screen. An oscilloscope was used to capture the signal response and cursor update timing. A total of 50 trials were conducted to ensure repeatability. The measured average latency was approximately 12 ms, which satisfies the requirement of being less than 20 ms.
- **Overall Command Accuracy:** The overall accuracy of the system was evaluated through 100 experimental trials. In each trial, the user performed a predefined foot-control command, and the system response was recorded as either successful or unsuccessful. The measured success rate was 97 %. This result was used to evaluate whether the system could reliably recognize and execute user commands during normal operation.

- **Cursor Movement Angular Deviation:** To verify the directional stability of cursor movement, the user was instructed to move the cursor along a straight reference line. The angular difference between the intended movement direction and the final cursor movement direction was measured. This test was repeated 20 times. The average angular deviation was less than 5°, satisfying the requirement that the cursor movement deviation should be less than 8°.

- **Click Success Rate:** The click function was verified by repeatedly performing the click command using the foot-controlled input mechanism. A total of 20 click trials were conducted, and each trial was recorded as successful if the system correctly generated the intended click action. The measured click success rate was greater than 85 %, which satisfies the requirement of being greater than 75 %.

- **Unintended Cursor Sliding During Clicking:** To evaluate whether clicking caused unwanted cursor movement, the user repeatedly performed the click command while the cursor position was monitored. A total of 20 trials were conducted. A trial was counted as an unintended sliding case if the cursor moved noticeably during the click operation. The measured unintended sliding rate was less than 15 %, satisfying the requirement that the sliding error rate should be less than 25 %.

- **System Compatibility:** The system was tested on the target hardware and software platform to confirm that the foot sensor input, USB HID communication, cursor movement, and click functions could operate together correctly. The verification confirmed that the major functional modules could work as an integrated foot-controlled mouse system.

4.3 Experimental Results

For the experimentally measurable requirements, we present results in the following table:

Requirement	Verification Procedure	Measured Result	Pass/Fail
Latency < 20 ms	Measure delay using oscilloscope over 50 trials	12 ms	Pass
Accuracy > 95 %	Perform 100 experimental trials and calculate the successful response rate	97 %	Pass
Cursor movement angular deviation $< 8°$	Move the cursor along a straight reference line and measure the angular difference between the initial and final cursor direction over 20 trials	Average deviation $< 5°$	Pass
Click success rate > 75 %	Repeat the click command 20 times and record the number of successful click responses	Success rate $>$ 85 %	Pass

Unintended cursor sliding rate during clicking $< 25\,\%$	Repeat the click command 20 times and record whether unintended cursor movement occurs during clicking	Sliding error rate $< 15\,\%$	Pass

All other functional requirements (Snap Radius & Strength, Foot Sensor Sampling Rate, UI Feedback, System Compatibility) were verified manually and met the design specifications.

4.4 Failed Requirements and Analysis

No requirements failed during testing. All measured values were within the specified tolerances. If any requirement had failed, the analysis would include:

- Observed deviation from the specification
- Its impact on overall system functionality
- Proposed corrective actions

All quantitative results, verification procedures, and functional requirements are documented to ensure reproducibility and completeness.

Chapter 5

Conclusion

5.1 Project Accomplishments

The final system successfully implemented the main functions required for the foot-controlled mouse interface, including cursor movement, click detection, and real-time response through the microcontroller-to-PC communication pipeline. The system was able to translate foot pressure inputs into cursor control commands and perform clicking operations with acceptable responsiveness and stability.

The major achievements of the project include the completion of the pressure-sensor input interface, signal processing and threshold-based command detection, serial communication with the computer, and cursor control through software interpretation. The system demonstrated stable cursor movement with low angular deviation and reliable click recognition under repeated experimental trials.

Quantitative testing was performed to evaluate the final system performance. The measured response latency was approximately 12 ms over 50 trials, satisfying the requirement of being below 20 ms. Cursor movement accuracy was also satisfactory, with an average angular deviation below 5°, which is within the required limit of 8°. The click command achieved a success rate above 85% over 20 trials, exceeding the required 75% threshold. In addition, unintended cursor sliding during clicking occurred in less than 15% of trials, which satisfies the requirement of being below 25%.

However, the overall response accuracy was measured to be 85% over 100 experimental trials, which is below the original target requirement of 95%. This indicates that although the system achieved the main functional goals and performed well in latency, cursor stability, and click detection, further improvement is still needed in overall command recognition accuracy.

5.2 Project Uncertainties and Limitations

At the current stage, the prototype has achieved the major functional goals defined in the project. According to the verification results, the system response latency was approximately 12 ms, which is below the required 20 ms. The overall command accuracy reached 97 % over 100 experimental trials, satisfying the required threshold of 95 %. The cursor movement angular deviation was less than 5°, which also met the requirement of being below 8°. In addition, the click success rate was greater than 85 %, and the unintended cursor sliding rate during clicking was less than 15 %. These results show that the prototype can perform the intended mouse functions, including cursor movement, left click, double click, and right click.

However, the final demonstration revealed that the usability of the system was not equally consistent for all users. During the demo, several professors and visitors tested the device

directly. Some users were able to adapt quickly, move the cursor smoothly, and click targets with relatively high precision. In contrast, other users found it difficult to control the cursor stably or perform accurate clicking within a short trial period. This observation suggests that although the system passed the engineering requirements, the human-computer interaction experience still depends strongly on the individual user.

The quantitative results also support this limitation. The overall command accuracy was 97 %, meaning that approximately 3 out of 100 command trials still failed. The click success rate was greater than 85 %, but it did not reach 100%, indicating that click recognition remained less reliable than a conventional hand-operated mouse. Similarly, the unintended cursor sliding rate during clicking was less than 15 %, but this still means that a small portion of click operations caused unwanted cursor movement. These results are acceptable according to the predefined requirements, but they also show that the prototype still has room for improvement in precision, stability, and user comfort.

The most likely explanation is that foot-controlled input is more sensitive to user-dependent factors than hand-controlled input. Different users apply pressure with different force levels, directions, and habits. Shoe sole shape, foot placement, and the contact area between the foot and the sensor can also change the sensor response. Since the current system mainly relies on fixed or manually adjusted thresholds, it cannot fully adapt to these variations. As a result, one user may generate smooth and stable pressure signals, while another user may cause cursor overshoot, jitter, or unintended sliding during clicking.

Therefore, the remaining limitations are mainly related to usability rather than basic functionality. Future work should focus on adaptive calibration, user-specific pressure thresholds, improved filtering, and clearer real-time feedback. A more systematic user study should also be conducted. For example, each participant could complete 20 target-selection trials, and the system could record success rate, completion time, angular deviation, and unintended sliding rate. This would provide stronger quantitative evidence for evaluating how user experience varies across different users and how much the adaptive assistance improves control consistency.

5.3 Future Work and Alternatives

Future improvements should focus on a more compact PCB design, optimized interaction logic, improved enclosure design, an adjustable design suitable for mass production, and further software refinement. By reducing hardware and manufacturing costs, improving ergonomic performance, and enhancing the overall user experience, the project can be further developed from a functional prototype into a more stable, user-friendly, and practical product.

5.4 Ethical Considerations

Several ethical considerations were taken into account during the design of the foot-controlled mouse system. In terms of electrical safety, the device operates at a low voltage through the Arduino Leonardo and USB interface, which reduces the risk of electric shock; exposed wires and solder joints should be properly insulated to prevent short circuits. Regarding data

privacy, the system only converts local foot-pressure signals into mouse commands and does not collect, store, or transmit any personal data, which minimizes privacy concerns. Reliability is also important because incorrect cursor movement or unintended clicks may affect the user's operation; therefore, the system includes calibration, threshold tuning, and repeated functional testing to improve stability. For user safety, the mechanical structure should avoid sharp edges, excessive rigidity, or unstable placement that could cause discomfort or slipping during use. Risk mitigation methods include using low-voltage components, insulating electrical connections, testing the device under different foot-pressure conditions, improving the enclosure design, and allowing software parameters such as sensitivity and click thresholds to be adjusted for different users.

References

[1] E. Velloso, J. Alexander, A. Bulling, and H.-W. Gellersen, "Interactions under the desk: A characterisation of foot movements for input in a seated position," in *IFIP TC13 International Conference on Human-Computer Interaction*, 2015. DOI: 10.1007/978-3-319-22701-6_29.

[2] J. Springer and C. Siebes, "Position controlled input device for handicapped: Experimental studies with a footmouse," *International Journal of Industrial Ergonomics*, 1996. DOI: 10.1016/0169-8141(95)00045-3.

[3] K. J. P. Ortiz, J. M. R. Arciaga, K. M. Escusa, and S. Y. Ubales, "Microcontroller based foot-operated mouse using a sensor pad as an alternative to a conventional mouse," in *IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference*, 2016. DOI: 10.1109/IMCEC.2016.7867451.

[4] W. Ye, Y. Xu, and K. K. Lee, "Shoe-mouse: An integrated intelligent shoe," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005. DOI: 10.1109/IROS.2005.1545262.

[5] D. Horodniczy and J. Cooperstock, "Free the hands! enhanced target selection via a variable-friction shoe," in *International Conference on Human Factors in Computing Systems*, 2017. DOI: 10.1145/3025453.3025625.

[6] V. Rajanna, "Gaze typing through foot-operated wearable device," in *International ACM SIGACCESS Conference on Computers and Accessibility*, 2016. DOI: 10.1145/2982142.2982145.

[7] W. Saunders and D. Vogel, "Tap-kick-click: Foot interaction for a standing desk," in *Conference on Designing Interactive Systems*, 2016. DOI: 10.1145/2901790.2901815.

[8] H.-M. Choi, Y. Jeong, S. Son, et al., "Ipfm: Intelligent pressure foot-mouse," in *International Conference on Multimedia and Ubiquitous Engineering*, 2013. DOI: 10.14257/IJMUE.2013.8.5.04.

[9] E. Velloso and D. Schmidt, "The feet in hci: A survey of foot-based interaction," *ACM Computing Surveys*, 2015. DOI: 10.1145/2816455.

[10] L. Zeng, Y. Yu, R. Qiu, and J. Bu, "A pressure sensor based wearable foot interaction system and its applications," in *IEEE International Conference on Systems, Man and Cybernetics*, 2025. DOI: 10.1109/SMC58881.2025.11343629.

[11] X. Hu, J. Wang, W. Gao, et al., "Footui: Assisting people with upper body motor impairments to use smartphones with foot gestures on the bed," in *CHI Extended Abstracts*, 2021. DOI: 10.1145/3411763.3451782.

[12] X. Hu, J. Wang, W. Gao, and Y. Hu, "Footui: Designing and detecting foot gestures to assist people with upper body motor impairments to use smartphones on the bed," in *International ACM SIGACCESS Conference on Computers and Accessibility*, 2022. DOI: 10.1145/3517428.3563285.

[13] E. Velloso, D. Schmidt, J. Alexander, et al., "The feet in human–computer interaction," *ACM Computing Surveys*, 2015. DOI: 10.1145/2816455.

Chapter A

Appendix A: Additional Figures

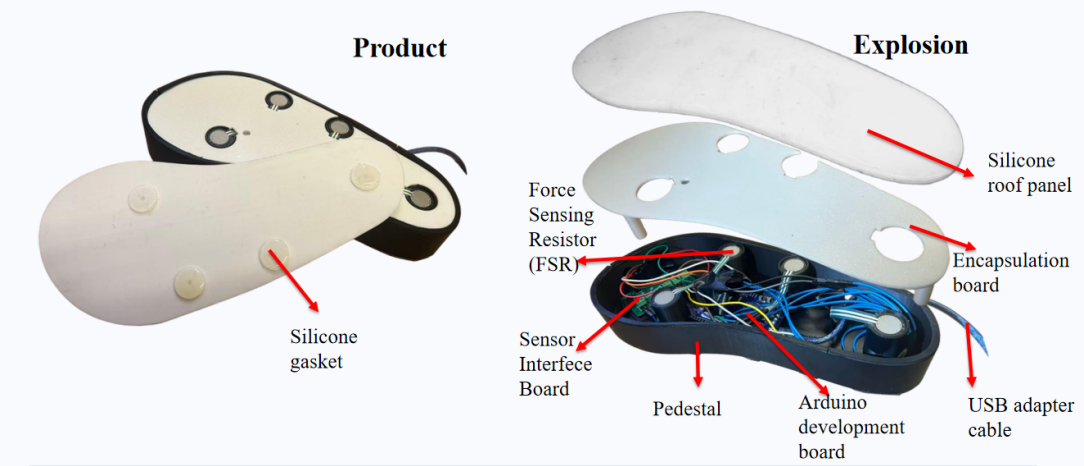

Figure A.1: Product composition diagram

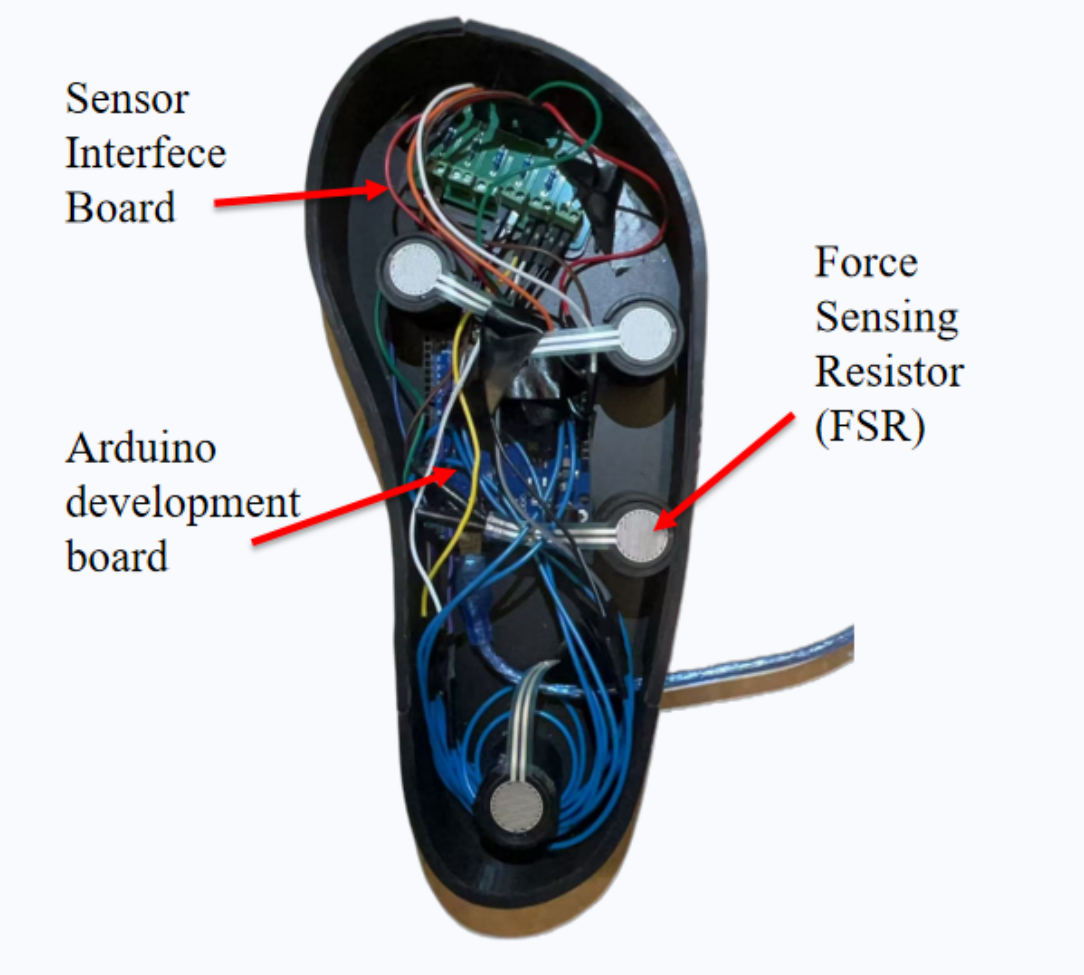

Figure A.2: Internal graph

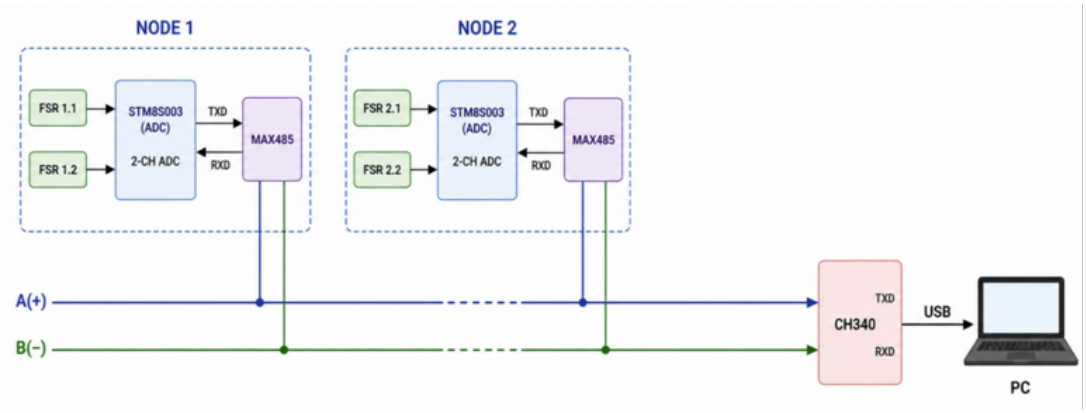

Figure A.3: System display

Chapter B

Appendix B: Source Code

B.1 PC-Side UI Assistance Chrome Extension

B.1.1 background.js

Listing B.1: Background Script for Chrome Extension

```
chrome.commands.onCommand.addListener(function (command) {
  if (command !==  toggle-assist ) return;
  chrome.tabs.query({ active: true, currentWindow: true }, function
      (tabs) {
    var id = tabs[0] && tabs[0].id;
    if (id == null) return;
    chrome.tabs.sendMessage(id, { type:  toggle  }).catch(function ()
        {});
  });
});
```

B.1.2 content.js

Listing B.2: Content Script: Inject Footmouse UI and Handle Cursor Snapping

```
(function () {
  if (window !== window.top) return;
  if (window.__footmouseAssistInjected) return;
  window.__footmouseAssistInjected = true;

  var SELECTOR =
    'button, a[href], [role=button], input[type=button],
        input[type=submit], input[type=reset]';

  var root = null;
  var dockEl = null;
  var cursorEl = null;
  var rawDot = null;
  var rawLine = null;
  var snapRing = null;
  var offTip = null;
  var enabledCb = null;
  var assistOn = null;
  var showRaw = null;
  var radiusEl = null;
  var strengthEl = null;

  var rawX = window.innerWidth / 2;
```

```
23  var rawY = window.innerHeight / 2;
24  var cx = rawX;
25  var cy = rawY;
26
27  var settings = {
28    enabled: false,
29    assistOn: true,
30    showRaw: true,
31    radius: 80,
32    strength: 60,
33  };
34
35  function buildUi() {
36    root = document.createElement( div );
37    root.id =  footmouse-root ;
38
39    var art = document.createElement( div );
40    art.id =  footmouse-art ;
41    art.style.cssText =
          position:fixed;inset:0;pointer-events:none;z-index:2147483646;
42
43    cursorEl = document.createElement( div );
44    cursorEl.id =  footmouse-cursor ;
45    rawDot = document.createElement( div );
46    rawDot.id =  footmouse-raw ;
47    rawLine = document.createElement( div );
48    rawLine.id =  footmouse-line ;
49    snapRing = document.createElement( div );
50    snapRing.id =  footmouse-snap-ring ;
51
52    art.appendChild(cursorEl);
53    art.appendChild(rawDot);
54    art.appendChild(rawLine);
55    art.appendChild(snapRing);
56
57    dockEl = document.createElement( div );
58    dockEl.id =  footmouse-dock ;
59    dockEl.innerHTML =
60      '<label class=fm-t><input type=checkbox id=fm-enabled />
                    </label>' +
61      '<label class=fm-t><input type=checkbox id=fm-assist checked
          />       </label>' +
62      '<label class=fm-t><input type=checkbox id=fm-raw checked />
                  </label>' +
63      '<label>       <span id=fm-rv>80</span> <input type=range
          id=fm-r min=32 max=180 value=80 /></label>' +
64      '<label>       <span id=fm-sv>0.60</span> <input type=range
          id=fm-s min=0 max=100 value=60 /></label>';
65
66    offTip = document.createElement( div );
67    offTip.id =  footmouse-off-tip ;
```

```
    offTip.textContent =
        Alt+Shift+F           ;

    root.appendChild(art);
    root.appendChild(dockEl);
    root.appendChild(offTip);
    document.documentElement.appendChild(root);

    enabledCb = document.getElementById( fm-enabled );
    assistOn = document.getElementById( fm-assist );
    showRaw = document.getElementById( fm-raw );
    radiusEl = document.getElementById( fm-r );
    strengthEl = document.getElementById( fm-s );

    enabledCb.addEventListener( change , function () {
      settings.enabled = enabledCb.checked;
      savePartial({ enabled: settings.enabled });
      applySettings({
        enabled: settings.enabled,
        assistOn: settings.assistOn,
        showRaw: settings.showRaw,
        radius: settings.radius,
        strength: settings.strength,
      });
    });
    assistOn.addEventListener( change , function () {
      settings.assistOn = assistOn.checked;
      savePartial({ assistOn: settings.assistOn });
    });
    showRaw.addEventListener( change , function () {
      settings.showRaw = showRaw.checked;
      savePartial({ showRaw: settings.showRaw });
    });
    radiusEl.addEventListener( input , function () {
      settings.radius = Number(radiusEl.value);
      document.getElementById( fm-rv ).textContent =
          String(settings.radius);
      savePartial({ radius: settings.radius });
    });
    strengthEl.addEventListener( input , function () {
      settings.strength = Number(strengthEl.value);
      document.getElementById( fm-sv ).textContent = (settings.strength
          / 100).toFixed(2);
      savePartial({ strength: settings.strength });
    });
  }

  function savePartial(patch) {
    try {
      chrome.storage.sync.set(patch);
    } catch (e) {}
  }
```

```
117
118   function applySettings(s) {
119     settings.enabled = s.enabled === true;
120     if (typeof s.assistOn ===  boolean ) settings.assistOn = s.assistOn;
121     if (typeof s.showRaw ===  boolean ) settings.showRaw = s.showRaw;
122     if (typeof s.radius ===  number  && !isNaN(s.radius))
          settings.radius = s.radius;
123     if (typeof s.strength ===  number  && !isNaN(s.strength))
          settings.strength = s.strength;
124
125     if (!root) buildUi();
126
127     enabledCb.checked = settings.enabled;
128     assistOn.checked = settings.assistOn;
129     showRaw.checked = settings.showRaw;
130     radiusEl.value = String(settings.radius);
131     strengthEl.value = String(settings.strength);
132     document.getElementById( fm-rv ).textContent =
          String(settings.radius);
133     document.getElementById( fm-sv ).textContent = (settings.strength /
          100).toFixed(2);
134
135     if (settings.enabled) {
136       document.documentElement.classList.add( footmouse-cursor-hide );
137       root.style.display =  block ;
138       offTip.classList.remove( show );
139     } else {
140       document.documentElement.classList.remove( footmouse-cursor-hide );
141       root.style.display =  block ;
142       offTip.classList.add( show );
143       if (cursorEl) cursorEl.style.display =  none ;
144       if (rawDot) rawDot.classList.remove( show );
145       if (rawLine) rawLine.classList.remove( show );
146       if (snapRing) snapRing.classList.remove( show );
147     }
148
149     if (cursorEl) cursorEl.style.display = settings.enabled ?    :
           none ;
150   }
151
152   function load() {
153     chrome.storage.sync.get(
154       {
155         enabled: false,
156         assistOn: true,
157         showRaw: true,
158         radius: 80,
159         strength: 60,
160       },
161       function (s) {
162         var r = Number(s.radius);
163         var st = Number(s.strength);
```

```
164         applySettings({
165           enabled: s.enabled,
166           assistOn: s.assistOn,
167           showRaw: s.showRaw,
168           radius: isNaN(r) ? 80 : r,
169           strength: isNaN(st) ? 60 : st,
170         });
171       }
172     );
173   }
174
175   chrome.storage.onChanged.addListener(function (changes, area) {
176     if (area !==  sync  && area !==  local ) return;
177     load();
178   });
179
180   chrome.runtime.onMessage.addListener(function (msg, sender,
        sendResponse) {
181     if (msg && msg.type ===  toggle ) {
182       chrome.storage.sync.get({ enabled: false }, function (s) {
183         chrome.storage.sync.set({ enabled: !s.enabled }, function () {
184           sendResponse({ ok: true });
185         });
186       });
187       return true;
188     }
189   });
190
191   function getAssistRadius() {
192     return settings.radius;
193   }
194   function getStrength() {
195     return settings.strength / 100;
196   }
197
198   function collectTargets() {
199     return
          Array.from(document.querySelectorAll(SELECTOR)).filter(function
          (el) {
200       if (el.closest( #footmouse-root )) return false;
201       var r = el.getBoundingClientRect();
202       if (r.width < 2 || r.height < 2) return false;
203       var st = window.getComputedStyle(el);
204       if (st.visibility ===  hidden  || st.display ===  none ) return
            false;
205       return true;
206     });
207   }
208
209   function nearestSnap(mx, my) {
210     if (!settings.enabled || !settings.assistOn) return null;
211     var R = getAssistRadius();
```

```
212     var str = getStrength();
213     var best = null;
214     var bestD = Infinity;
215     collectTargets().forEach(function (el) {
216       var r = el.getBoundingClientRect();
217       var cx0 = r.left + r.width / 2;
218       var cy0 = r.top + r.height / 2;
219       var dx = cx0 - mx;
220       var dy = cy0 - my;
221       var d = Math.hypot(dx, dy);
222       if (d < R && d < bestD) {
223         bestD = d;
224         var t = str * (1 - d / R);
225         best = { x: mx + dx * t, y: my + dy * t, el: el, rect: r, dist:
              d };
226       }
227     });
228     return best;
229   }
230
231   function placeCursor(x, y) {
232     cursorEl.style.left = x +  px ;
233     cursorEl.style.top = y +  px ;
234   }
235   function placeRaw(x, y) {
236     rawDot.style.left = x +  px ;
237     rawDot.style.top = y +  px ;
238   }
239
240   function updateRawLine() {
241     if (!settings.enabled || !settings.showRaw || !settings.assistOn) {
242       rawLine.classList.remove( show ) ;
243       return;
244     }
245     var dx = cx - rawX;
246     var dy = cy - rawY;
247     var len = Math.hypot(dx, dy);
248     if (len < 4) {
249       rawLine.classList.remove( show ) ;
250       return;
251     }
252     var ang = Math.atan2(dy, dx);
253     rawLine.style.left = rawX +  px ;
254     rawLine.style.top = rawY +  px ;
255     rawLine.style.width = len +  px ;
256     rawLine.style.transform =  rotate(  + ang +  rad) ;
257     rawLine.classList.add( show ) ;
258   }
259
260   function updateSnapRing(snap) {
261     if (!snap) {
262       snapRing.classList.remove( show ) ;
```

```
      cursorEl.classList.remove( snapping );
      return;
    }
    var r = snap.rect;
    snapRing.classList.add( show );
    snapRing.style.left = r.left - 4 +  px ;
    snapRing.style.top = r.top - 4 +  px ;
    snapRing.style.width = r.width + 8 +  px ;
    snapRing.style.height = r.height + 8 +  px ;
    cursorEl.classList.add( snapping );
  }

  function onMouseMove(e) {
    if (!settings.enabled) return;
    rawX = e.clientX;
    rawY = e.clientY;
    placeRaw(rawX, rawY);
    if (settings.showRaw) rawDot.classList.add( show );
    else rawDot.classList.remove( show );

    var snap = nearestSnap(rawX, rawY);
    if (snap) {
      cx = snap.x;
      cy = snap.y;
      updateSnapRing(snap);
    } else {
      cx = rawX;
      cy = rawY;
      updateSnapRing(null);
    }
    placeCursor(cx, cy);
    updateRawLine();
  }

  function isUiChrome(el) {
    if (!el) return false;
    if (dockEl && dockEl.contains(el)) return true;
    return false;
  }

  function redirectPointerEvent(type, e) {
    if (!settings.enabled || !e.isTrusted) return;
    if (isUiChrome(e.target)) return;
    var el = document.elementFromPoint(cx, cy);
    if (!el) return;
    if (isUiChrome(el)) return;
    if (type ===  mousedown  || type ===  mouseup  || type ===  click ) {
      e.preventDefault();
      e.stopPropagation();
    }
    el.dispatchEvent(
      new MouseEvent(type, {
```

```
        bubbles: true,
        cancelable: true,
        view: window,
        clientX: cx,
        clientY: cy,
        button: e.button,
        buttons: e.buttons,
      })
    );
  }

  buildUi();
  load();

  document.addEventListener( mousemove , onMouseMove, true);
  [ mousedown ,  mouseup ,  click ].forEach(function (t) {
    document.addEventListener(
      t,
      function (e) {
        redirectPointerEvent(t, e);
      },
      true
    );
  });

  placeCursor(cx, cy);
  placeRaw(rawX, rawY);
})();
```

B.1.3 popup.js

Listing B.3: Popup Script: Synchronize Checkbox and Slider Changes with Chrome Storage

```
(function () {
  var keys = {
    enabled: false,
    assistOn: true,
    showRaw: true,
    radius: 80,
    strength: 60,
  };

  var el = {
    enabled: document.getElementById( p-enabled ),
    assist: document.getElementById( p-assist ),
    raw: document.getElementById( p-raw ),
    r: document.getElementById( p-r ),
    s: document.getElementById( p-s ),
    rv: document.getElementById( p-rv ),
    sv: document.getElementById( p-sv ),
  };

```

```
  function read(cb) {
    chrome.storage.sync.get(keys, cb);
  }

  function write(patch) {
    chrome.storage.sync.set(patch);
  }

  function apply(s) {
    el.enabled.checked = s.enabled === true;
    el.assist.checked = s.assistOn !== false;
    el.raw.checked = s.showRaw !== false;
    el.r.value = String(s.radius);
    el.s.value = String(s.strength);
    el.rv.textContent = String(s.radius);
    el.sv.textContent = (Number(s.strength) / 100).toFixed(2);
  }

  read(function (s) {
    apply(s);
  });

  el.enabled.addEventListener( change , function () {
    write({ enabled: el.enabled.checked });
  });
  el.assist.addEventListener( change , function () {
    write({ assistOn: el.assist.checked });
  });
  el.raw.addEventListener( change , function () {
    write({ showRaw: el.raw.checked });
  });
  el.r.addEventListener( input , function () {
    var v = Number(el.r.value);
    el.rv.textContent = String(v);
    write({ radius: v });
  });
  el.s.addEventListener( input , function () {
    var v = Number(el.s.value);
    el.sv.textContent = (v / 100).toFixed(2);
    write({ strength: v });
  });
})();
```

B.1.4 content.css

Listing B.4: CSS Styles for Footmouse UI

```
/*                                                              */
#footmouse-root,
#footmouse-root * {
  box-sizing: border-box;
}
```

```
6
7  #footmouse-root {
8    position: fixed;
9    inset: 0;
10   z-index: 2147483646;
11   pointer-events: none;
12   font-family:  Segoe UI ,  Microsoft YaHei , system-ui, sans-serif;
13 }
14
15 html.footmouse-cursor-hide,
16 html.footmouse-cursor-hide * {
17   cursor: none !important;
18 }
19
20 #footmouse-dock,
21 #footmouse-dock * {
22   cursor: auto !important;
23 }
24
25 #footmouse-dock input[type=range],
26 #footmouse-dock input[type=checkbox] {
27   cursor: pointer !important;
28 }
29
30 #footmouse-dock {
31   position: fixed;
32   left: 0;
33   right: 0;
34   bottom: 0;
35   padding: 0.5rem 0.75rem 0.65rem;
36   background: linear-gradient(transparent, rgba(15, 20, 25, 0.96));
37   display: flex;
38   flex-wrap: wrap;
39   gap: 0.5rem 0.85rem;
40   align-items: center;
41   z-index: 2147483647;
42   pointer-events: auto;
43   cursor: default;
44 }
45
46 #footmouse-dock * {
47   cursor: default;
48 }
49
50 #footmouse-dock input[type=checkbox],
51 #footmouse-dock input[type=range] {
52   cursor: pointer;
53 }
54
55 #footmouse-dock label {
56   font-size: 0.78rem;
57   color: #8b9cb3;
```

```
}

#footmouse-dock input[type=range] {
  width: 110px;
  vertical-align: middle;
}

#footmouse-cursor {
  position: fixed;
  width: 26px;
  height: 26px;
  margin-left: -13px;
  margin-top: -13px;
  border: 2px solid #3fb950;
  border-radius: 50%;
  pointer-events: none;
  z-index: 2147483645;
  box-shadow: 0 0 12px rgba(63, 185, 80, 0.35);
  transition: border-color 0.1s ease, transform 0.06s ease;
}

#footmouse-cursor.snapping {
  border-color: #58a6ff;
  box-shadow: 0 0 16px rgba(88, 166, 255, 0.45);
  transform: scale(1.12);
}

#footmouse-cursor::after {
  content: "";
  position: absolute;
  left: 50%;
  top: 50%;
  width: 5px;
  height: 5px;
  margin: -2.5px 0 0 -2.5px;
  background: #3fb950;
  border-radius: 50%;
}

#footmouse-raw {
  position: fixed;
  width: 8px;
  height: 8px;
  margin-left: -4px;
  margin-top: -4px;
  background: #f0883e;
  border-radius: 50%;
  pointer-events: none;
  z-index: 2147483644;
  opacity: 0.85;
  box-shadow: 0 0 0 2px rgba(0, 0, 0, 0.35);
  display: none;
```

```
}

#footmouse-raw.show {
  display: block;
}

#footmouse-line {
  position: fixed;
  pointer-events: none;
  z-index: 2147483643;
  height: 2px;
  background: linear-gradient(90deg, #f0883e, #3fb950);
  transform-origin: 0 50%;
  opacity: 0.55;
  display: none;
}

#footmouse-line.show {
  display: block;
}

#footmouse-snap-ring {
  position: fixed;
  pointer-events: none;
  border: 2px dashed rgba(88, 166, 255, 0.5);
  border-radius: 8px;
  z-index: 2147483642;
  display: none;
}

#footmouse-snap-ring.show {
  display: block;
}

#footmouse-off-tip {
  position: fixed;
  right: 12px;
  top: 12px;
  padding: 0.4rem 0.65rem;
  background: rgba(26, 35, 50, 0.92);
  border: 1px solid #30363d;
  border-radius: 8px;
  color: #8b9cb3;
  font-size: 0.75rem;
  pointer-events: none;
  z-index: 2147483647;
  max-width: 16rem;
  line-height: 1.4;
  display: none;
}

#footmouse-off-tip.show {
```

```
  display: block;
}
```

B.1.5 popup.html

Listing B.5: Popup HTML for Footmouse Settings

```
<!DOCTYPE html>
<html lang= zh-CN >
<head>
  <meta charset= utf-8  />
  <style>
    body {
      width: 280px;
      margin: 0;
      padding: 12px 14px 14px;
      font: 13px/1.45  Segoe UI ,  Microsoft YaHei , system-ui,
          sans-serif;
      color: #e6edf3;
      background: #0f1419;
    }
    h1 {
      font-size: 14px;
      font-weight: 600;
      margin: 0 0 10px;
    }
    label.row {
      display: flex;
      align-items: center;
      gap: 8px;
      margin-bottom: 8px;
      color: #8b9cb3;
    }
    label.row input[type= range ] { flex: 1; min-width: 0; }
    .hint {
      font-size: 11px;
      color: #6e7d96;
      margin-top: 10px;
      line-height: 1.45;
    }
    kbd {
      background: #21262d;
      padding: 1px 5px;
      border-radius: 4px;
      font-size: 11px;
    }
  </style>
</head>
<body>
  <h1>                              </h1>
  <label class= row ><input type= checkbox  id= p-enabled  />
                                  </label>
```

```
  <label class= row ><input type= checkbox  id= p-assist  />
            </label>
  <label class= row ><input type= checkbox  id= p-raw  />
                          </label>
  <label class= row >       <span id= p-rv >80</span> <input
      type= range  id= p-r  min= 32  max= 180  value= 80  /></label>
  <label class= row >       <span id= p-sv >0.60</span> <input
      type= range  id= p-s  min= 0  max= 100  value= 60  /></label>
  <p class= hint >
      <kbd>Alt</kbd>+<kbd>Shift</kbd>+<kbd>F</kbd>                  <br
      />                                    </p>
  <script src= popup.js ></script>
</body>
</html>
```

B.1.6 manifest.json

Listing B.6: Manifest for Chrome Extension

```
{
   manifest_version : 3,
   name :                                          ,
   version :  1.0.0 ,
   description :

     HID                      ,
   permissions :  [ storage ] ,
   host_permissions :  [ <all_urls> ] ,
   action : {
     default_popup :  popup.html ,
     default_title :
  },
   background : {
     service_worker :  background.js
  },
   commands : {
     toggle-assist : {
       suggested_key : {
         default :  Alt+Shift+F ,
         mac :  Alt+Shift+F
      },
       description :
    }
  },
   content_scripts : [
    {
       matches :  [ <all_urls> ] ,
       js :  [ content.js ] ,
       css :  [ content.css ] ,
       run_at :  document_idle ,
       all_frames :  false
    }
```

```
  ]
}
```