TECHNICAL DATASHEET

NORA

Modular Physical Assistant with Embedded Intelligence

Model:	NORA-V1
Document version:	DS-NORA-1.0
Release date:	May 30th, 2025
Author:	Alberto Marquillas
License:	MIT License
Contact:	`https://github.com/AlbertoMarquillas`
Organization:	Intelligent Prototyping Lab (IPL), Barcelona

Contents

1. General Overview

The NORA system is a modular, intelligent physical assistant designed for embedded operation in personal, educational, and experimental environments. It integrates voice recognition, computer vision, agent-based behavior logic, and physical mobility, all supported by a fully local architecture that eliminates reliance on cloud infrastructure.

The device is built around a decentralized control model, where software modules operate semi-independently and communicate through a central Finite State Machine (FSM) controller. This allows for predictable behavior patterns, easy debugging, and a scalable logic core. NORA operates in real-time, responding to vocal commands, visual stimuli, and environmental events through FSM-driven transitions and adaptive agents.

In its standard configuration, NORA includes a primary processor board (e.g., Raspberry Pi 4 Model B), a camera module, a microphone array, actuators (wheels or servos), and a local storage interface functioning as a NAS. Additional modules, such as an emotion engine, external sensor interfaces, and network-discoverable subsystems, can be enabled depending on the application.

NORA is fully customizable, and each subsystem—voice, vision, motion, memory, and logic—can be independently developed, replaced, or expanded. This makes it suitable for research projects, robotics curricula, personal productivity, or interactive installations where user feedback and autonomy are required.

The system is developed with a privacy-first philosophy. All inference, classification, memory storage, and behavior logic occur within the device boundary. No data is transmitted externally unless explicitly configured.

Key design principles include:

- **Modularity:** Each hardware and software block is isolated, versioned, and extensible.
- **Determinism:** The FSM core ensures traceable behavior and reproducible state transitions.
- **Local Execution:** No cloud dependency is required for primary functionality.
- **Voice and Visual Fusion:** Commands and recognition pipelines can be cross-validated.
- **File-Centric Memory:** Data, logs, and recorded events are stored in a structured, user-accessible format.

Multiple modes of interaction are supported, including:

- Voice-only operation for environments where screens are not desirable.
- Web-based GUI control via a local dashboard, including sensor display and file access.
- Autonomous mode with no user input, responding to ambient stimuli or recurring routines.

The current firmware version supports real-time speech processing, audio feedback, gesture recognition, movement planning, and file-based recall. Future releases will include integration with reinforcement-based learning agents, improved navigation modules, and wireless sensor auto-discovery (EdgeLink protocol).

NORA is distributed under the MIT License and is suitable for academic, prototyping, or personal enhancement purposes.

1.1 Product Introduction

The NORA system is a modular, embedded physical assistant designed to operate autonomously in human-centric environments. Developed as a general-purpose experimental platform, NORA integrates perception, logic, memory, and actuation into a single extensible system.

At its core, NORA functions as a standalone, programmable device capable of voice-based interaction, visual analysis, motion control, and memory retention. Its architecture is optimized for local processing, modular scalability, and deterministic behavior through a finite state machine (FSM)-driven logic system.

NORA does not rely on cloud infrastructure for operation. All critical functionalities—including speech recognition, system control, and data logging—are executed locally. This design ensures data privacy, fast response times, and full control by the user.

The hardware platform is based on off-the-shelf components, typically built around a Raspberry Pi 4 Model B board, with USB or CSI camera support, microphone input, audio output, general-purpose I/O, and local storage through USB or SSD interfaces. A fully documented API and hardware abstraction layer allow for easy integration of new sensors, actuators, and behavior modules.

NORA is intended for use in the following scenarios:

- Development and prototyping of embedded intelligence systems.
- Personal assistance in private, offline environments.
- Education in robotics, control systems, and human-computer interaction.
- Testbed for adaptive agents and multimodal control interfaces.
- Research into real-time decision-making and privacy-preserving AI systems.

The system can be deployed in multiple configurations: as a stationary voice- and vision-enabled assistant, as a mobile platform capable of physical navigation, or as an integrated node in a sensor-rich smart environment. In all cases, NORA emphasizes transparency, modularity, and embedded control.

This document describes the functional characteristics, internal architecture, and operational details of NORA version 1.0, including hardware recommendations, subsystem descriptions, and integration options.

1.2 Target Applications

NORA is designed as a versatile and modular embedded system suitable for a broad range of applications where localized intelligence, multimodal interaction, and deterministic behavior are required. The system's internal modularity, open design, and local processing capabilities make it particularly well suited for the following domains:

Educational Environments

- Robotics laboratory exercises and embedded systems coursework.
- Training platforms for FSM design, voice interaction, and multimodal integration.
- Student projects in physical computing, AI, or human-machine interaction.

Research and Prototyping

- Human-aware robotics, embedded agents, and dialogue systems.
- On-device AI research in vision, audio, or reinforcement learning.
- Experimental deployments for multimodal user studies.

Personal Assistance and Home Integration

- Local smart assistant for room-level interaction.
- Data logger and personal memory tool with offline NAS storage.
- Mobile assistant for intradomestic navigation and object detection.

Interactive Installations and Public Spaces

- Kiosks or assistive robots in museums, galleries, or libraries.
- Demonstrators for educational fairs, exhibitions, or science outreach.
- Emotion-aware installations for ambient feedback or guidance.

Testbed for Advanced Control Systems

- Finite State Machine experimentation and behavior graph validation.
- Real-time sensor fusion using voice, image, and environmental inputs.
- Controlled evaluation of system transitions under dynamic conditions.

Privacy-Conscious Environments

- Use cases where no data should leave the local network.
- Offline interaction systems for children, patients, or vulnerable users.
- Regulatory environments (e.g., GDPR-restricted regions) where cloud-free operation is required.

1.3 Key Features Summary

NORA integrates a wide range of features commonly required in autonomous assistant systems. The following table summarizes the primary functionalities supported in the standard configuration of NORA-V1.

Feature	Enabled by Default	Description / Notes
Voice Recognition (offline)	Yes	Real-time speech-to-text using local acoustic models; no internet required.
Text-to-Speech Synthesis	Yes	Converts system responses or reminders into natural voice output.
FSM-Controlled Behavior Logic	Yes	Core finite state machine engine for deterministic transitions.
Vision-Based Tracking	Yes	Uses USB/CSI camera to detect movement, face, or object targets.
File-Based Memory (NAS)	Yes	All logs, notes, and user memories stored locally; accessible via web GUI.
Local Web Dashboard (GUI)	Yes	React-based interface for viewing logs, states, and interacting with the system.
Mobility Control System	Optional	Configurable motor drivers (e.g., L298N) and motion planning engine.
Sensor Expansion via EdgeLink	Optional	Detect and integrate external devices (ESP32, Pi, etc.) via MQTT or mDNS.
Energy Monitoring	Optional	Tracks battery level, triggers low-power behavior modes, optional solar input.
Emotional State Estimation	Planned	Internal agent model to associate sensory inputs with emotional context.
Personal Diary (voice/text)	Yes	Entries recorded manually or by command; stored in human-readable Markdown.
Offline Operation (cloud-free)	Yes	All core functionalities designed to work without any internet connection.

Table 1: *

Table 1-1: Summary of Functional Features in NORA-V1

1.4 System Architecture

NORA is structured as a decentralized embedded system composed of five primary functional domains: voice processing, vision analysis, control logic, mobility subsystem, and memory management. These domains are managed by a centralized Finite State Machine (FSM)

engine, which orchestrates transitions based on system events, user commands, or sensor inputs.

The architecture emphasizes local processing and modular separation. Each subsystem is encapsulated in its own software service and communicates with the central FSM controller via internal events. This structure allows each block to be developed, debugged, and extended independently.

All critical functionalities are performed on-device. Communication with external agents, such as sensors or remote clients, is managed through MQTT and/or HTTP interfaces, depending on the use case. A local dashboard built in React enables real-time monitoring and manual control, while the FSM and agents maintain autonomous operation in the background.

The following diagram illustrates the high-level architecture of the system.

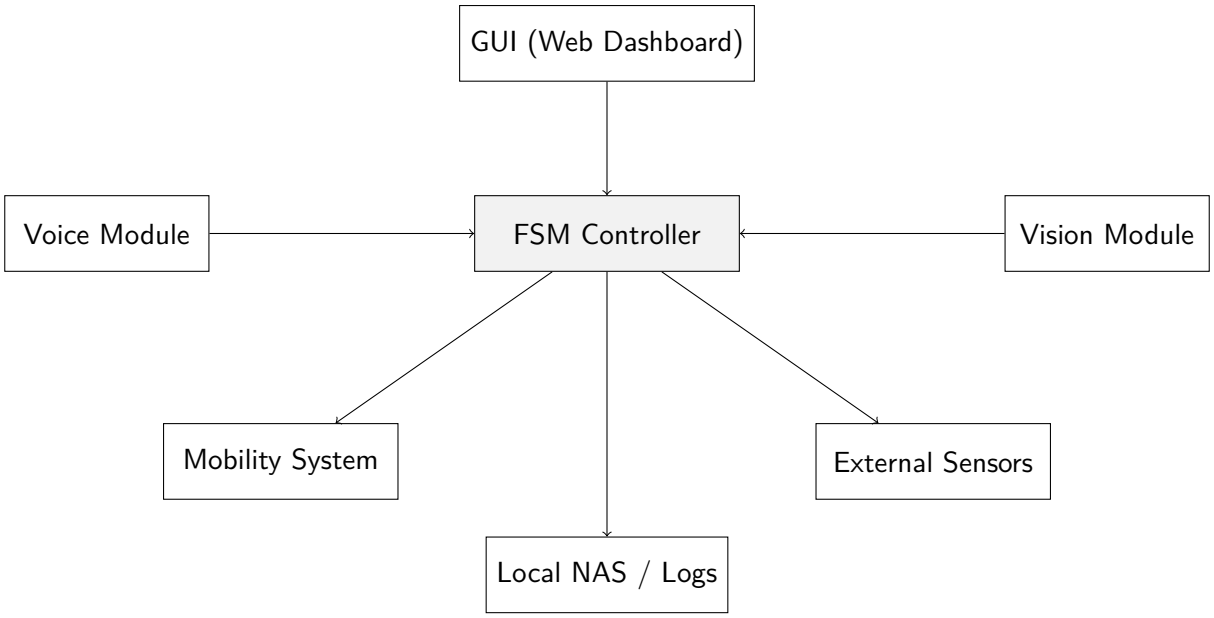

Figure 1: *

Figure 1-1: System architecture overview of NORA

1.5 Subsystem Overview

The NORA platform is composed of five primary functional subsystems, each responsible for a critical aspect of system operation. These are:

1. Voice Interface
2. Vision Interface
3. FSM Control Logic
4. Mobility Subsystem
5. NAS and Memory Structure

Each subsystem is implemented as an independent software module, with clearly defined interfaces, event triggers, and execution contexts. Subsystems can be developed and tested in isolation, and are fully configurable during deployment. They are designed to operate both in direct response to user input and as part of autonomous behavior loops managed by the FSM controller.

The following subsections (1.5.1 through 1.5.5) describe the internal role, interfaces, dependencies, and characteristics of each subsystem. The modular design allows for flexible scaling depending on available hardware, power constraints, or desired functionality.

In standard operation, these modules are executed concurrently and communicate via shared memory, messaging queues, or FSM event triggers. Their design allows the system to remain operational even if one or more subsystems are temporarily disabled, ensuring a degree of fault tolerance and graceful degradation.

Subsystem modularity also enables partial deployment configurations, such as:

- Headless voice-only assistant (no vision or mobility).
- Stationary node with NAS logging and FSM-based reactive logic.
- Mobile unit with navigation and gesture interaction, but without GUI.

These modes can be defined at compile time or through runtime configuration files, depending on the build target.

1.5.1 Voice Interface

The Voice Interface module is responsible for all audio-based interaction in the NORA system, including speech recognition, voice command processing, and text-to-speech (TTS) synthesis. It serves as the primary input modality in non-visual configurations and complements the vision system when multimodal interaction is active.

The subsystem is composed of the following internal components:

- **Audio Input Handler:** Captures microphone data in real-time using a pulseaudio or ALSA interface. Sampling rate is typically 16 kHz, mono-channel.
- **Speech Recognition Engine:** Performs offline transcription using an embedded model (e.g., Vosk, Whisper.cpp, or PocketSphinx). The system supports both continuous listening and activation word modes.
- **Command Parser:** Maps transcribed text to internal events or FSM triggers. Can differentiate between direct commands (e.g., "go forward") and indirect queries (e.g., "what time is it?").
- **Text-to-Speech Synthesizer:** Converts system-generated responses into spoken audio using local TTS engines (e.g., espeak-ng, Festival, or PicoTTS).
- **FSM Interface:** Sends event flags and parsed content to the FSM controller for evaluation and execution.

Operational Modes:

- **Passive listening:** Commands are accepted only after an activation keyword is detected (e.g., "NORA").

- **Continuous listening:** Open command recognition with periodic resets and silence detection to prevent flooding.
- **Manual trigger:** External hardware (button, GUI) activates a single recognition window.

Dependencies and Resources:

- Compatible with USB or onboard audio input.
- CPU usage depends on model; real-time performance achievable on quad-core ARM-based boards.
- Requires no internet connection; all models and grammars are stored locally.

Event Flow Example:

1. Microphone detects speech input and digitizes signal.
2. Audio is passed to the recognition engine.
3. Resulting text is parsed and matched to a known intent.
4. FSM is notified with the intent as an event.
5. FSM triggers a response or state transition.
6. Optional TTS engine vocalizes the system's response.

1.5.2 Vision Interface

The Vision Interface module enables real-time visual perception capabilities within the NORA system. It is responsible for acquiring image data, performing basic analysis, extracting relevant features, and emitting high-level events to the FSM controller.

It operates as a continuously running service that processes frames from a connected camera device, applying a configurable set of detection pipelines depending on the operational mode and current FSM state.

Core Components:

- **Camera Driver:** Interface for USB (UVC) or CSI-based cameras. Captures video frames at standard resolution (typically 640x480 @ 30 fps).
- **Frame Processor:** Applies preprocessing steps such as grayscale conversion, Gaussian filtering, and resizing.
- **Feature Extractor:** Analyzes frames using methods such as:
  - Face detection (Haar cascades, DNN)
  - Motion estimation (frame differencing)
  - Contour or shape recognition

  - Object detection (YOLOv5-lite or MobileNet SSD, optional)

- **Event Mapper:** Converts extracted features or triggers (e.g., "face detected", "movement on left") into FSM events.

Operational Modes:

- **Surveillance Mode:** Monitors for presence or movement and triggers alert events.
- **Face Interaction Mode:** Detects face proximity, orientation or expressions for interaction adaptation.
- **Gesture Mode (Optional):** Uses trained models to recognize hand gestures for input (under development).
- **Tracking Mode:** Maintains lock on a moving object or face and provides coordinates to the mobility system.

Integration and Outputs:

- Events are sent to the FSM in symbolic form (e.g., `face_near`, `movement_left`, `no_target`).
- Visual overlays are optionally rendered for debugging via OpenCV.
- Diagnostic data (e.g., FPS, resolution, active detectors) can be logged locally.

Dependencies and Performance:

- Requires OpenCV 4.x or compatible image processing library.
- Processing load is adjustable via frame skip rate and resolution.
- Can operate in parallel with the Voice Interface under standard thermal limits on Raspberry Pi 4.

Event Flow Example:

1. Camera captures frame at 30 fps.
2. Frame processor normalizes the image.
3. Face detector identifies one or more faces.
4. Event `face_detected` is sent to the FSM.
5. FSM transitions to `interaction` state and enables voice response.

1.5.3 FSM Control Logic

The FSM (Finite State Machine) Control Logic module is the core of NORA's decision-making and behavioral control framework. It governs the system's operation by managing discrete states and defining allowed transitions based on internal and external events.

Each subsystem in NORA communicates with the FSM controller via structured events. These events can be triggered by voice commands, visual detections, sensor readings, timeouts, or internal conditions. The FSM controller evaluates each event against its current state and transition table, applying the corresponding action and updating the system state accordingly.

FSM Controller Components:

- **State Table:** Defines all legal states, including idle, listening, navigating, responding, logging, etc.
- **Transition Matrix:** Maps each event in each state to a target state and optional action.
- **Guard Conditions:** Optional logical conditions (boolean expressions) that must be met for a transition to occur.
- **Event Queue:** Receives and prioritizes incoming events from all active subsystems.
- **Action Dispatcher:** Executes system-level actions (e.g., move, speak, record) as part of a transition.

State Properties:

- States are exclusive; only one active state at a time.
- Each state may have associated entry and exit actions.
- Some states are interruptible (e.g., listening), while others are protected (e.g., critical execution).

Examples of Core States:

- `IDLE` – Waiting for interaction.
- `LISTENING` – Active voice recognition.
- `RESPONDING` – Synthesizing or executing response.
- `TRACKING` – Vision tracking active.
- `MOVING` – Motion system in progress.
- `LOGGING` – Writing data to persistent memory.

Example Transition:

Current State	Event	Next State	Action
IDLE	voice_wakeword	LISTENING	Enable microphone
LISTENING	voice_command_found	RESPONDING	Trigger TTS output
TRACKING	face_lost	IDLE	Stop motors
IDLE	sensor_motion	TRACKING	Start vision module

Implementation Details:

- Implemented in Python as a standalone controller class.
- Stateless transition definitions stored as dictionaries or JSON for portability.
- Supports hot-reload of FSM structure without full system reboot.
- Fully decoupled from subsystems; operates via event abstraction.

Design Advantages:

- Predictable behavior and execution path.
- Easy debugging with transition tracing.
- Modular and transparent logic suitable for safety-critical prototyping.

1.5.4 Mobility Subsystem

The Mobility Subsystem provides NORA with locomotion capabilities, enabling it to navigate its physical environment, respond to spatial commands, and autonomously approach targets or charging stations.

This subsystem is composed of motor drivers, wheel encoders (optional), motion control algorithms, and interaction logic with the central FSM Controller. It receives movement instructions through high-level commands (e.g., "go to X", "follow object", "dock") and translates them into low-level PWM control.

Physical Setup:

- **Motors:** 2× DC gear motors with encoders (optional)
- **Drivers:** Dual H-Bridge (e.g., L298N or ODrive for BLDC)
- **Power:** Independent 12 V rail for traction system
- **Platform:** Differential drive chassis

Motion Modes:

- `MANUAL` – Controlled remotely via voice or interface
- `AUTONOMOUS` – Moves according to internal FSM state (e.g., follow object, explore area)
- `PATROL` – Predefined cyclic route

- `DOCKING` – Seeks and aligns to charging station
- `STOPPED` – Motors disabled, system in static mode

Motion Commands (abstracted):

- `MOB_GO_FORWARD` – Move forward a fixed distance
- `MOB_ROTATE_LEFT` – Turn on axis counter-clockwise
- `MOB_FOLLOW_TARGET` – Adjust motion based on vision tracking
- `MOB_STOP` – Immediately stop all motors

Sensors and Feedback:

- Ultrasonic sensors for front/back obstacle detection
- Optional rotary encoders for position estimation
- IMU (optional) for heading stabilization
- Docking beacon (IR or magnetic guide)

FSM Integration:

Each motion command is triggered by FSM events. Example:

FSM State	Event	Next State	Action
IDLE	sensor_motion	TRACKING	Enable motors
TRACKING	vision_target_found	MOVING	MOB_FOLLOW_TARGET
MOVING	target_lost	IDLE	MOB_STOP
IDLE	low_battery	DOCKING	MOB_SEEK_DOCK

Safety Features:

- Emergency stop pin (hardware interrupt)
- Timeout watchdog for motor commands
- Battery-level constraints integrated in FSM logic

Implementation Notes:

- Motor PWM control implemented using Raspberry Pi GPIO + external driver
- Motion logic managed in an independent Python thread with shared state
- Docking logic supports autonomous recharging and state feedback

1.5.5 Emotional Engine

The Emotional Engine is a conceptual and software subsystem designed to simulate internal affective states within NORA. These emotional states influence both behavior and multimodal expressions, including lighting, posture, vocal tone, and movement patterns.

Rather than using pure randomness, NORA bases its emotional changes on external stimuli, user interaction patterns, and internal system metrics such as battery level or temperature.

Emotional States (core set):

- `NEUTRAL` – Default, idle state
- `CURIOUS` – Triggered by new input or sensory detection
- `HAPPY` – Positive feedback or social engagement
- `TIRED` – Low battery or long uptime
- `ALERT` – Motion detected or unexpected event
- `SAD` – Lack of interaction or shutdown request

State Transitions:

Emotional state is updated through a weighted evaluation of inputs over time. A sample transition matrix:

Trigger	Current State	Next State
No interaction for ¿10 min	NEUTRAL	SAD
User voice detected	NEUTRAL or SAD	HAPPY
New NFC tag detected	NEUTRAL	CURIOUS
Low battery warning	Any	TIRED
Movement in environment	Any	ALERT
Charging dock reached	TIRED	NEUTRAL

Expression Outputs:

- **RGB Lighting:** Internal LED color or pulsing frequency
- **Voice Modulation:** Pitch and rhythm variation
- **Posture and Movement:** Tilting head, small motions
- **Facial Animation:** (optional) Display-based facial expressions
- **Log Entry:** Each emotional shift is stored in the system diary

Implementation:

- Python module running as a background agent
- State stored in shared memory accessible by GUI and Voice modules

- Configurable thresholds and reaction profiles
- Exposes API for external modules to query or modify current state

Interaction Example:

- FSM triggers event: ‘user_speaks‘
- Emotional engine updates state to `HAPPY`
- LED fades to warm yellow, speech becomes more energetic
- Diary logs: ‘14:23 — Detected friendly voice, changed state to HAPPY‘

1.5.6 Memory and Diary System

The Memory and Diary subsystem enables NORA to store meaningful user experiences, states, and interaction histories in a structured and revisitable manner. It combines contextual metadata, emotional state tracking, voice annotations, and timestamps to build an internal “life log” of its operation.

Core Features:

- **Memory Logs:** Each relevant event (voice command, sensor detection, emotional shift) is logged with context.
- **Daily Diary:** Summarizes key events and emotional states per day.
- **Voice Notes:** Users can dictate messages to be stored as diary entries.
- **Markdown Export:** Textual entries can be exported to human-readable ‘.md‘ files.
- **NAS Integration:** All entries are saved within NORA’s local NAS and accessible via GUI.

Event Format Example:

```
2025-06-01 18:32 | [EVENT] NFC tag "MyKeys" detected
2025-06-01 18:33 | [EMOTION] Shifted to CURIOUS
2025-06-01 18:40 | [VOICE] "Remind me to call Alex tomorrow"
2025-06-01 18:41 | [DIARY] Entry created: "Reminder set: call Alex"
```

Voice Command Examples:

- “Take a note: I’m feeling tired today.”
- “Remember that the red folder belongs to math.”
- “Show me my diary from last week.”

Data Organization:

- `/storage/diary/YYYY-MM-DD.md` — Text diary for each day
- `/storage/logs/emotions.log` — Chronological emotional states
- `/storage/voice-notes/YYYY-MM-DD_hhmm.ogg` — Optional audio notes
- `/storage/reminders.json` — Persistent reminders/events

Internal Architecture:

- Python-based logging daemon
- Receives input from FSM, Emotional Engine, Voice Module
- Automatically rotates and archives logs daily
- Provides REST endpoint for frontend visualization

Access and Review:

- Through GUI $\rightarrow$ "Diary" section (calendar + list view)
- By voice $\rightarrow$ "Show my memories from today"
- Remotely via NAS interface (read-only access)

FSM & Agent-Based Control

NORA's behavior is governed by a combination of a centralized Finite State Machine (FSM) and modular intelligent agents. This architecture allows for predictable control logic (FSM) while supporting reactive and adaptive behavior through autonomous subsystems (agents).

1.6.1 Global FSM Architecture

The main FSM defines the high-level operational modes of NORA. Each state encapsulates behaviors and activates relevant modules.

- **IDLE** – Minimal activity; awaiting input
- **ACTIVE** – Full attention, processing input and executing tasks
- **FOLLOW** – Locomotion mode following a user or object
- **PATROL** – Autonomous movement through known areas
- **DOCKING** – Navigating to charging station
- **SLEEP** – Power-saving and ambient monitoring

Each state transition is triggered by events (e.g., voice commands, sensor triggers, internal thresholds) and validated by guard conditions.

1.6.2 Event Processing Flow

Events are received via:

- **Voice input**
- **Sensor detection**
- **Time-based triggers**
- **Remote commands**

All events are processed by the `FSMController`, which evaluates transitions, updates the state, and propagates signals to agents.

1.6.3 Internal Agents

Agents are lightweight services responsible for modular tasks. They receive context from the FSM and act independently or in cooperation.

Agent Name	Role
`NavigationAgent`	Path planning, obstacle avoidance, and destination arrival logic
`EmotionAgent`	State evaluation, expression rendering, and memory contribution
`VisionAgent`	Real-time object/person detection and spatial localization
`VoiceAgent`	Recognition of commands, synthesis, and contextual query processing
`EnergyAgent`	Monitoring power levels and triggering docking behavior
`MemoryAgent`	Logging events, diary management, and log access control

1.6.4 Coordination Mechanism

Agents communicate via a shared context bus and notification dispatcher. Each module can:

- Emit signals (e.g., `low_battery`, `human_detected`)
- Subscribe to events (e.g., `state_changed: FOLLOW`)
- Modify the shared world model

The FSM remains the single source of truth for global state, while agents provide decentralized intelligence.

1.6.5 Real-Time Feedback Loop

The FSM-agent system enables tight control loops. For example:

1. The camera detects an obstacle.
2. The `VisionAgent` emits an `obstacle_detected` signal.
3. The `NavigationAgent` updates the route and notifies the FSM.
4. The FSM transitions from `PATROL` to `IDLE`.
5. The system emits a visual alert (e.g., LED state change).

This loop operates with low latency and supports both deterministic logic and adaptive behavior.

1.7 Modes of Operation

NORA supports multiple operational modes designed to adapt its functionality, responsiveness, and power consumption to various user contexts and environmental conditions. Each mode activates or restricts certain subsystems and behaviors according to the current scenario or user command.

1. **Normal Mode**
   Default operational configuration. All subsystems are active: FSM control, voice and vision interfaces, mobility, sensors, and memory logging. Suitable for interactive environments.
2. **Night Mode**
   Visual outputs are dimmed, LEDs reduced or disabled, and voice synthesis muted. Only essential sensors remain active. Designed for low-disturbance operation during rest hours.
3. **Follow Mode**
   Activates motion and vision systems to maintain proximity to a target (typically a user). FSM prioritizes trajectory tracking and obstacle avoidance. Activated by command or trigger.
4. **Silent Mode**
   Voice output is disabled. Interaction occurs through visual indicators or GUI only. Recommended for quiet or shared spaces.
5. **Maintenance Mode**
   Disables automatic behaviors. Allows manual control of motors, sensors, and interfaces. Used for calibration, testing, or hardware access.
6. **Debug Mode**
   Enables real-time internal logging of events, transitions, and sensor values. FSM remains active. Intended for development or troubleshooting.

7. **Charging Mode**
   Activated when docked or externally powered. Limits processing load and disables mobility. GUI may display charge status and logs system temperature/battery voltage.

8. **Offline Mode**
   Disables all network interfaces. Data remains strictly local. Ideal for privacy-restricted or disconnected environments.

9. **NAS Mode**
   Prioritizes file storage and transfer operations. Voice/vision modules may be disabled. NAS services (e.g., Samba) run at elevated priority.

10. **Learning Mode**
    Passive observation mode. Events are logged, but no active responses are generated. Useful for data collection, user modeling, or agent retraining.