Coding Conventions for Software Team

This document outlines standard coding conventions to ensure consistency, maintainability, and collaboration across the project.

1 General Guidelines

- Write clean, readable, and well-commented code.
- Follow a modular approach: break down features into reusable functions and components.
- Ensure that all code changes are thoroughly tested before merging into the main branch.
- Adhere to branch naming conventions as outlined in the main project README.

2 File and Directory Structure

- Organize files by feature or subsystem.
- Use clear and descriptive folder names.

Example directory structure:

```
proto-0/
|-- motor_control/          # Code for motor control logic
|-- sensors_io/             # Sensor management and communication
|-- feedback_control/       # Haptic and other feedback systems
|-- tests/                  # Testing frameworks for system modules
|-- docs/                   # Documentation and design notes
```

3 Naming Conventions

Files and Folders

- Use `SCREAMING_SNAKE_CASE` for file and folder names: `MOTOR_CONTROL.py`
- Avoid spaces or special characters in filenames.
- Include file version number at the end: `ESP32_MOTORCONTROL_V1.py`
- General format: `BOARD_FUNCTION_PROJECT_V#`
- Example: `ESP32_MOTORCONTROL_PROTO0_V3`

Variables and Constants

- Use `snake_case` for variable names: `motor_speed`
- Use `UPPER_SNAKE_CASE` for constants: `MAX_MOTOR_SPEED`

Functions and Methods

- Use `snake_case` for function and method names: `initialize_motor()`

Classes

- Use `PascalCase` for class names: `MotorController`

Branch Names

- Use the format: `proto-0/feature-description`
- Example: `proto-0/motor-control`

4 Commenting and Documentation

- Use comments to explain complex logic.
- Single-line comment style:

```
# This is a single-line comment
```

Function docstring example:

```
def calculate_motor_speed(speed: int) -> int:
    """
    Calculate the motor speed based on input speed value.
    Args:
        speed (int): Desired speed input.
    Returns:
        int: Adjusted motor speed.
    """
    return speed * 2
```

Maintain all documentation in the `/docs/` folder for system design, API references, and testing.

5 Code Formatting

- Use tabs for indentation.
- Keep lines under 80 characters when possible.

6 Git Commit Messages

- Use concise and descriptive commit messages.
- Format:

```
[module name] Brief description of change
```

Example:

```
[motor_control] Added PID controller for motor speed stabilization
```

7 Testing and Validation

- Write unit tests for all functions.
- Organize tests in the `/tests/` directory.
- Ensure all tests pass before submitting pull requests.

8 Pull Requests

- Provide a clear title and description of changes.
- Software Lead and/or Director will review.
- Ensure the code follows all conventions before requesting a review.

9 Error Handling

- Include appropriate error handling mechanisms.
- Avoid silent failures; always log errors where possible.

Example:

```
try:
    motor_speed = calculate_motor_speed(input_speed)
except ValueError as e:
    print(f”Invalid␣motor␣speed:␣{e}”)
```

This document will be updated as the project evolves. Please review and adhere to these conventions for all contributions to the project.