Exoskeleton Control System

System Design Document

Version 1.0

December 13, 2025

Table of Contents

1. Executive Summary 3

2. System Architecture 4

3. Core Components 5

4. Execution Pipeline 8

5. Data Model 10

6. Design Patterns and Principles 11

7. Performance Considerations 12

8. Safety and Robustness 13

9. Extension and Customization 14

10. Testing and Validation 15

11. Deployment Considerations 16

12. Future Enhancements 17

13. File Organization 18

14. Conclusion 19

1. Executive Summary

This document describes the architecture and design of an intelligent exoskeleton control system that enables real-time locomotion state estimation and adaptive assistance. The system integrates inertial measurement units (IMUs), machine learning classification, sensor fusion, and finite state machine control to provide seamless support for human mobility.

1.1 Purpose

The system aims to detect and respond to different locomotion modes (walking, standing, stair climbing) in real-time to provide appropriate exoskeleton assistance while ensuring user safety and natural movement patterns.

1.2 Key Features

- Real-time locomotion classification using Support Vector Machine (SVM)
- Multi-IMU sensor fusion with Kalman filtering for accurate joint angle estimation
- Finite State Machine (FSM) for smooth state transitions with confirmation thresholds
- Modular architecture supporting multiple public datasets (USC-HAD, HuGaDB)
- Configurable pipeline parameters through centralized configuration

2. System Architecture

2.1 Architecture Overview

The system follows a layered architecture with clear separation of concerns:

Layer	Components
Configuration	ExoConfig.m - Centralized system parameters
Data Acquisition	ImportData.m, PrepareTrainingData.m, Dataset loaders
Feature Extraction	Features.m - Time and frequency domain features
Sensor Fusion	FusionKalman.m - Kalman filter for orientation estimation
Classification	Classifier.m, StateEstimator.m, RealtimeFsm.m
Training & Testing	TrainSvmBinary.m, TestPipelinePerformance.m
Execution	RunExoskeletonPipeline.m - Main execution script

2.2 Data Flow

The system processes data through the following pipeline:

1. **Raw IMU Data Acquisition:** Accelerometer and gyroscope data from back, left hip, and right hip sensors (100 Hz sampling rate)
2. **Windowing:** Data segmented into 1-second windows with 50% overlap (50 samples step size)
3. **Feature Extraction:** Mean magnitude, variance, and dominant frequency computed for each window
4. **Classification:** SVM classifier predicts locomotion state (walking vs standing)
5. **State Estimation:** FSM validates predictions using confirmation threshold (3 consecutive predictions)
6. **Sensor Fusion:** Kalman filter fuses accelerometer and gyroscope data for joint angle estimation
7. **Control Command Generation:** Commands sent to exoskeleton actuators based on confirmed state

3. Core Components

3.1 Configuration Management (ExoConfig.m)

Centralizes all system parameters to enable easy tuning without modifying core algorithms.

3.1.1 Key Parameters

Parameter	Default Value	Description
FS	100 Hz	IMU sampling rate
WINDOW_SIZE_S	1.0 seconds	Analysis window duration
STEP_SIZE_S	0.5 seconds	Window overlap (50%)
FSM_CONFIRMATION	3 predictions	State transition threshold
ACCEL_NOISE	0.01	Kalman accelerometer noise
GYRO_NOISE	0.005	Kalman gyroscope noise

3.2 Data Acquisition Layer

3.2.1 ImportData.m

Responsible for loading raw IMU data from various sources and normalizing it to a consistent format.

Functionality:

- Reads CSV files containing accelerometer and gyroscope data
- Validates data integrity (sample count consistency, field presence)
- Returns structured data with back, hipL, and hipR sensor readings
- Supports both real-time sensor streams and pre-recorded datasets

3.2.2 PrepareTrainingData.m

Processes multiple trials from public datasets to create training/testing sets for the SVM classifier.

Processing Pipeline:

- Loads multiple USC-HAD trials using LoadUSCHAD.m
- Maps activity labels to binary classification (walking vs non-walking)
- Applies sliding window segmentation
- Extracts features using Features.m for each window
- Aggregates feature vectors and labels for training

3.2.3 Dataset Support

Dataset	Sensors	Activities
USC-HAD	Single IMU (waist-mounted)	12 activities (6 locomotion, 6 static)
HuGaDB	Multiple IMUs (left/right thighs)	8 activities including gait variations

3.3 Feature Extraction (Features.m)

Converts raw IMU data windows into discriminative features for classification.

Extracted Features:

8. **Mean Acceleration Magnitude:** Captures overall movement intensity. Walking typically shows higher values than standing.
9. **Variance of Acceleration Magnitude:** Measures movement regularity. Walking exhibits higher variance due to gait cycle.
10. **Dominant Frequency:** Extracted via FFT on vertical (Z-axis) acceleration. Walking shows characteristic 1-2 Hz pattern corresponding to step frequency.

Implementation Details:

- Magnitude calculated as Euclidean norm of 3-axis acceleration
- FFT computed on single-sided spectrum with DC component removed
- Features optimized for computational efficiency (real-time capable)
- Gyroscope data reserved for future enhancement

3.4 Sensor Fusion (FusionKalman.m)

Implements Kalman filtering to fuse accelerometer and gyroscope measurements for accurate orientation and joint angle estimation.

3.4.1 Filter Initialization

The initializeFilters function creates two independent IMU filter objects (one for back sensor, one for hip sensor) with optimized noise parameters.

Key Parameters:

- Sample Rate: 100 Hz (matches IMU data rate)
- Accelerometer Noise: 0.01 (tuned for exoskeleton dynamics)
- Gyroscope Noise: 0.005 (lower than accelerometer due to higher precision)
- Reference Frame: East-North-Up (ENU) convention

3.4.2 Joint Angle Computation

The estimateAngle function computes relative joint angles from orientation quaternions.

Process:

11. Convert quaternion orientations to Euler angles (ZYX sequence)
12. Extract pitch angle (sagittal plane rotation) for both segments
13. Compute hip flexion angle as pitch difference
14. Convert from radians to degrees for interpretability

Performance Target: Root Mean Square Error (RMSE) < 5 degrees for joint angle estimation

3.5 Classification Layer

3.5.1 Classifier.m - SVM Implementation

Binary Support Vector Machine classifier for locomotion mode detection.

Configuration:

- Kernel: Radial Basis Function (RBF) for non-linear decision boundaries
- Standardization: Enabled (normalizes features to zero mean, unit variance)
- Classes: 0 (standing), 1 (walking)
- Training via TrainSvmBinary.m using USC-HAD dataset

3.5.2 StateEstimator.m - State Validation

Wrapper around the SVM classifier that handles feature extraction and prediction for single windows.

Functionality:

- Accepts raw IMU window data (acceleration, gyroscope)
- Calls Features.m to extract feature vector
- Invokes trained SVM model for prediction
- Returns predicted state (0 or 1)

3.5.3 RealtimeFsm.m - Finite State Machine

Implements a two-state FSM with confirmation threshold to prevent spurious state transitions.

States:

- **STATE_STANDING (0):** User is stationary, exoskeleton locked
- **STATE_WALKING (1):** User is ambulatory, exoskeleton provides gait assistance

Transition Logic:

15. If prediction matches current state: reset counter, maintain state
16. If prediction differs: increment counter
17. If counter reaches threshold (3): execute transition and reset counter
18. Generate control command based on new state

Control Commands:

- **Command 0:** Lock exoskeleton joints (standing mode)
- **Command 1:** Initiate walking gait assistance (walking mode)

4. Execution Pipeline

4.1 Training Phase (TrainSvmBinary.m)

Prepares and trains the binary SVM classifier using public dataset trials.

Training Steps:

19. Load configuration parameters from ExoConfig
20. Call PrepareTrainingData to generate feature matrix and label vector
21. Train SVM using fitcsvm with RBF kernel and standardization
22. Evaluate model performance using cross-validation
23. Save trained model to results/Binary_SVM_Model.mat

4.2 Real-time Execution (RunExoskeletonPipeline.m)

Main execution script that simulates real-time exoskeleton control using recorded data.

Pipeline Flow:

24. 1. Initialization:

```
cfg = ExoConfig();

load(cfg.FILE.SVM_MODEL, 'trainedModel');

[fuse_back, fuse_hipL] = initializeFilters(cfg.FS);
```

25. 2. Data Loading:

```
[back, hipL, hipR] = ImportData(activity_folder);
```

26. **3. Real-time Loop (window-by-window processing):**

- Extract current window of IMU data
- Update Kalman filters with new measurements
- Compute joint angle via estimateAngle
- Predict locomotion state via StateEstimator
- Update FSM and get control command
- Log results (state, angle, command)

27. **4. Post-processing:**

- Generate time-series plots of state transitions
- Visualize joint angle trajectories
- Save results to structured file

4.3 Performance Testing (TestPipelinePerformance.m)

Validates system performance across multiple test scenarios and generates performance metrics.

Test Metrics:

- Classification accuracy (percentage of correct predictions)
- State transition latency (time from movement change to FSM transition)
- Joint angle RMSE (compared to ground truth if available)
- Processing time per window (computational efficiency)
- False positive/negative rates for state transitions

5. Data Model

5.1 IMU Data Structure

Standardized format for sensor data used throughout the system:

```
imu_data = struct(
  'acc',  [Nx3 double],  % Acceleration [m/s^2] (X, Y, Z)
  'gyro', [Nx3 double]   % Angular velocity [rad/s] (X, Y, Z)
);
```

5.2 Sensor Array Structure

Three-IMU configuration for comprehensive body segment tracking:

```
sensor_array = struct(
  'back', imu_data,  % Trunk/torso sensor
  'hipL', imu_data,  % Left hip/thigh sensor
  'hipR', imu_data   % Right hip/thigh sensor
);
```

5.3 Feature Vector Format

Three-dimensional feature space for classification:

```
features = [mean_magnitude, variance_magnitude, dominant_frequency]
% Size: [1 x 3] double
```

5.4 System State Representation

Complete state vector for real-time monitoring:

```
system_state = struct(
  'timestamp',        double,    % Time in seconds
  'fsm_state',        {0,1},     % Current FSM state
  'predicted_label',  {0,1},     % SVM prediction
  'hip_angle',        double,    % Joint angle [degrees]
  'control_command',  {0,1},     % Actuator command
  'confirmation_count', int      % FSM counter
);
```

6. Design Patterns and Principles

6.1 Modular Architecture

Each component has a single, well-defined responsibility with clear interfaces:

- **Separation of Concerns:** Data acquisition, feature extraction, classification, and fusion are independent modules
- **Loose Coupling:** Components communicate through standardized data structures
- **High Cohesion:** Related functionality grouped within modules

6.2 Configuration-Driven Design

All tunable parameters centralized in ExoConfig.m:

- Single source of truth for system parameters
- Easy parameter tuning without code changes
- Consistent parameters across all modules
- Version control of parameter changes

6.3 Data Pipeline Pattern

Sequential processing stages with clear data transformations:

```
Raw IMU → Windows → Features → Predictions → States → Commands
```

Each stage is stateless and deterministic, enabling:

- Easy testing and validation of individual stages
- Parallel processing opportunities
- Clear debugging and error isolation

6.4 State Machine Pattern

FSM manages system behavior with explicit states and transitions:

- Predictable behavior through defined state transitions
- Hysteresis through confirmation threshold prevents oscillation
- Easy extension to additional states (e.g., stairs, running)
- Clear logging of state changes for debugging

6.5 Strategy Pattern for Datasets

Dataset-specific loaders implement common interface:

- LoadUSCHAD.m and LoadHuGaDB.m provide consistent output format
- Easy addition of new datasets without modifying pipeline
- Dataset-specific quirks handled in loaders

7. Performance Considerations

7.1 Real-time Constraints

System must meet strict timing requirements for safe exoskeleton operation:

Metric	Target	Rationale
Processing latency	< 100 ms	Human gait cycle response
Window update rate	2 Hz (0.5 s)	50% overlap ensures smooth tracking
State transition delay	1.5 s (3 windows)	Balance between safety and responsiveness

7.2 Computational Efficiency

Optimization strategies for embedded deployment:

- **Minimal feature set:** Only 3 features for fast computation
- **Single-pass FFT:** Frequency analysis on Z-axis only
- **Incremental Kalman:** Filter state persists between iterations
- **Vectorized operations:** MATLAB optimized array operations

7.3 Memory Management

Efficient memory usage for continuous operation:

- Circular buffer for IMU data (fixed size)
- No dynamic memory allocation in real-time loop
- Pre-allocated arrays for feature vectors
- Periodic logging to disk to prevent memory overflow

8. Safety and Robustness

8.1 Error Handling

Multiple layers of error detection and recovery:

- **Input Validation:** Check for missing data, incorrect dimensions, NaN values
- **Sensor Fault Detection:** Identify stuck or saturated sensors
- **Graceful Degradation:** Fallback to safe state (standing/locked) on errors
- **Logging:** Comprehensive error logging for diagnostics

8.2 State Machine Safeguards

Mechanisms to prevent unsafe state transitions:

- **Confirmation Threshold:** Requires 3 consecutive predictions (1.5 seconds) before transition
- **Debouncing:** Prevents rapid oscillation between states
- **State History:** Tracks recent state transitions for anomaly detection
- **Emergency Stop:** Immediate transition to safe state on critical errors

8.3 Sensor Fusion Robustness

Kalman filter provides noise rejection and outlier handling:

- Tuned noise parameters balance responsiveness and stability
- Complementary fusion of accelerometer (low frequency) and gyroscope (high frequency)
- Automatic correction for sensor drift over time
- Outlier rejection through measurement validation gates

9. Extension and Customization

9.1 Adding New Locomotion Modes

System designed for easy extension to additional activities:

28. Update ExoConfig.m to define new state constants
29. Collect training data for new activity
30. Retrain classifier with multi-class SVM (or neural network)
31. Extend FSM with new states and transition rules
32. Define control commands for new state
33. Test and validate new behavior

9.2 Integrating Additional Sensors

Architecture supports sensor expansion:

- **Pressure sensors:** Add ground contact detection
- **EMG sensors:** Incorporate muscle activation patterns
- **Force sensors:** Measure joint torques
- **GPS/Magnetometer:** Add outdoor navigation capabilities

Integration Steps:

34. Extend data import functions to include new sensor readings
35. Augment feature extraction with sensor-specific features
36. Update sensor fusion algorithm if applicable
37. Retrain classifier with expanded feature set

9.3 Alternative Classification Algorithms

Modular design allows classifier replacement:

- **Random Forest:** Better for non-linear decision boundaries
- **Neural Network:** Can learn complex temporal patterns
- **LSTM:** Sequential modeling of gait cycles
- **Ensemble Methods:** Combine multiple classifiers for robustness

Replacement requires modifying TrainSvmBinary.m and Classifier.m while maintaining input/output interface.

10. Testing and Validation

10.1 Unit Testing

Test scripts in tests/ directory validate individual components:

- **acquisition_test.m:** Validates data import and preprocessing
- **classification_test.m:** Tests classifier accuracy on labeled data
- **fusion_test.m (to be added):** Validates Kalman filter performance

10.2 Integration Testing

TestPipelinePerformance.m validates end-to-end system:

- Tests complete pipeline with multiple activity scenarios
- Validates state transitions occur at expected times
- Measures processing latency and computational efficiency
- Generates performance reports and visualizations

10.3 Validation Datasets

System validated against established benchmarks:

Dataset	Usage
USC-HAD	Primary training and testing dataset for binary classification
HuGaDB	Multi-sensor validation and gait pattern analysis
Custom Raw Data	Real-world testing with project-specific sensor configuration

11. Deployment Considerations

11.1 Hardware Requirements

11.1.1 Minimum Specifications

- **Processor:** Dual-core ARM Cortex-A9 or equivalent (>1 GHz)
- **Memory:** 512 MB RAM minimum, 1 GB recommended
- **Storage:** 100 MB for system, 1 GB for logging
- **IMU Sensors:** 3x 6-DOF IMUs (accelerometer + gyroscope), 100 Hz capable

11.1.2 Sensor Specifications

- **Accelerometer:** ±16g range, <1% linearity
- **Gyroscope:** ±2000 dps range, <2% linearity
- **Communication:** I2C or SPI interface
- **Power:** Low-power mode support for battery operation

11.2 Software Dependencies

11.2.1 MATLAB Requirements

- MATLAB R2020b or later
- Signal Processing Toolbox
- Statistics and Machine Learning Toolbox
- Sensor Fusion and Tracking Toolbox

11.2.2 Embedded Deployment

For embedded systems, consider:

- **MATLAB Coder:** Generate C/C++ code from MATLAB
- **Embedded Coder:** Optimize for specific hardware
- **Fixed-Point Designer:** Convert to fixed-point for efficiency
- **ROS Integration:** Deploy as ROS nodes for robotic systems

11.3 Calibration Procedures

11.3.1 Sensor Calibration

38. **Zero-G Calibration:** Collect static data to determine sensor bias
39. **Alignment Calibration:** Ensure sensor coordinate frames are aligned
40. **Scale Factor:** Verify measurement accuracy against known inputs

11.3.2 User Calibration

41. **Initial Pose:** User stands still for filter initialization (5 seconds)
42. **Range of Motion:** Collect data for user-specific gait patterns
43. **Personalization:** Optional adaptive tuning based on initial sessions

12. Future Enhancements

12.1 Short-term Improvements

- **Multi-class Classification:** Extend beyond binary to detect stairs, running, sitting
- **Adaptive Thresholding:** Dynamic FSM confirmation based on confidence
- **Gyroscope Features:** Incorporate angular velocity features for improved accuracy
- **Cloud Logging:** Remote monitoring and diagnostics

12.2 Medium-term Enhancements

- **Deep Learning:** CNN or LSTM for automatic feature learning
- **Gait Phase Detection:** Identify swing/stance phases for precise control
- **Terrain Detection:** Adapt assistance based on surface type
- **Energy Optimization:** Minimize actuator power consumption

12.3 Long-term Vision

- **Intent Recognition:** Predict user intentions from movement patterns
- **Multi-user Learning:** Federated learning across exoskeleton fleet
- **Rehabilitation Tracking:** Long-term gait improvement monitoring
- **Brain-Computer Interface:** Direct neural control integration

13. File Organization

13.1 Directory Structure

The project follows a standardized directory layout:

```
AutomationForExoskeleton/
├── config/              # System configuration
│   └── ExoConfig.m
├── data/                # All datasets
│   ├── public/          # Public datasets (USC-HAD, HuGaDB)
│   ├── raw/             # Project-specific raw data
│   ├── processed/       # Processed/cleaned data
│   └── interim/         # Intermediate processing outputs
├── src/                 # Source code
│   ├── acquisition/     # Data import modules
│   ├── features/        # Feature extraction
│   ├── fusion/          # Sensor fusion
│   └── classification/  # Classification & FSM
├── scripts/             # Execution scripts
│   ├── RunExoskeletonPipeline.m
│   ├── TrainSvmBinary.m
│   └── TestPipelinePerformance.m
├── tests/               # Unit tests
├── models/              # Trained models
├── results/             # Output files
└── docs/                # Documentation
```

13.2 Naming Conventions

- **Functions:** PascalCase (e.g., ImportData.m, Features.m)
- **Scripts:** PascalCase with descriptive names (e.g., RunExoskeletonPipeline.m)
- **Variables:** camelCase (e.g., windowSize, trainedModel)
- **Constants:** UPPER_SNAKE_CASE (e.g., FS, WINDOW_SIZE)
- **Files:** Descriptive names matching primary function/script name

14. Conclusion

This system design document provides a comprehensive overview of the exoskeleton control system architecture, enabling developers and AI systems to understand and extend the codebase effectively.

14.1 Key Architectural Strengths

- **Modularity:** Clear separation of concerns enables independent development and testing
- **Configurability:** Centralized parameters simplify tuning and adaptation
- **Extensibility:** Well-defined interfaces support new sensors, classifiers, and states
- **Robustness:** Multiple safety layers ensure reliable operation
- **Performance:** Optimized for real-time embedded deployment

14.2 Development Guidelines

When extending or modifying the system:

44. Always update ExoConfig.m for new parameters
45. Maintain consistent data structure formats
46. Add unit tests for new components
47. Document all API changes
48. Validate real-time performance after modifications
49. Follow established naming conventions
50. Update this design document for significant changes

14.3 References

- USC-HAD Dataset: Zhang, M. and Sawchuk, A. A. (2012)
- HuGaDB Dataset: Chereshnev, R. and Kertész-Farkas, A. (2017)
- MATLAB Sensor Fusion Toolbox Documentation
- Support Vector Machines: Cortes, C. and Vapnik, V. (1995)
- Kalman Filtering Theory: Kalman, R. E. (1960)

--- End of Document ---