Code Analysis

1. Data Loading and Pre-processing

- **Data Loading (`rdsamp`)**: It loads raw accelerometer data (likely from a standard format like WFDB/PhysioNet) from the specified file (`'Data/CO003.dat'`), obtaining the signal (Acc, angular velocity), sampling frequency (Fs), and time vector (tm).

- **Coordinate Transformation (`algo_Moe_Nilssen`)**: It applies a function to correct the raw signals (Vertical, Medial-Lateral, Antero-Posterior) for sensor **tilt** and **gravity (g)**. This results in **linear accelerations** (aAP , aML , aV) that are independent of the sensor's orientation and only represent body movement (i.e., the *dynamic* acceleration components).

2. Analysis Parameters

- **Window Size**: The analysis is performed on **1-second windows** (`window_sec = 1`).

- **Bout Definition**: A detected period of walking must have a minimum duration of **60 seconds** (`min_bout_duration_sec`).

- **Microbreak Tolerance**: A **1-second** tolerance (`microbreak_tolerance_sec`) is set to smooth out short, non-walking pauses (microbreaks).

- **Thresholds**: Two thresholds are defined for classifying activity in a 1-second window:

    - **SMA Threshold:** 0.135 (`threshold_SMA`). [169–176. doi:10.1016/j.medengphy.2013.06.005]
    - **Energy Threshold**: 0.05 (`threshold_en`).

3. Filtering and Classifications

- **High-Pass Filtering (`filtfilt`)**: The dynamic acceleration signals (aV ,aML ,aAP) are **high-pass filtered** at **0.5 Hz** to remove any remaining low-frequency drift or movement that isn't characteristic of rhythmic activities like walking.

- **Windowing and Metric Calculation**: The filtered signals are divided into contiguous **1-second windows**.

    - **SMA Calculation**: For each window, the **SMA (Signal Magnitude Area)** is calculated as the mean of the sum of the absolute values of the three axes (V, ML, AP).

$$\text{SMA} = \frac{1}{N} \sum_{k=1}^{N} (|a_{V,k}| + |a_{ML,k}| + |a_{AP,k}|)$$

    - **Energy Calculation**: The **Power Spectral Density (PSD)** is estimated using **`pwelch`** on the **AP axis** acceleration in each window. The 'Energy' metric is the area under the PSD curve (**Trapezoidal rule, `trapz`**) in the specific **walking frequency band** of **0.5 Hz to 3 Hz**.

- **Classification (`walking`)**: A window is classified as **"walking"** if **either** the SMA is greater than its threshold **OR** the Energy is greater than its threshold (an **OR logic**).

4. Bout Identification and Final Result

- **Microbreak Cleaning (`medfilt1`)**: A **median filter** is applied to the raw `walking` classification. This effectively **fills microbreaks** of up to 1 second within what should be a continuous walking period, smoothing the result.

- **Bout Filtering**: The smoothed walking periods are then analyzed to find continuous segments (bouts). Only segments that are longer than the **minimum duration of 60 seconds** are considered **valid walking bouts**.

- **Final Output**:

    - It reports the **number of valid walking bouts** found.

    - It calculates and displays the **final percentage of total analyzed time** that was spent walking in valid bouts.

What the Code is Doing in Simple Terms

The code is a program that analyzes accelerometer data to figure out **how much time a person spent walking in long, continuous periods**.

1. It cleans up the raw motion data to isolate the actual body movement.

2. It calculates two different measures of activity (**SMA** and **Energy**) for every single second of the recording.

3. If either measure crosses a set threshold, that second is marked as **active**.

4. It *cleans up* the results by ignoring very short pauses (like hesitations) within walking.

5. Finally, it counts up only the walking periods that last for **over 60 seconds** and calculates the total percentage of time spent in these long walking sessions.

```
clc; clear all; close all;

record_name = 'Data/CO003.dat';

disp(['Caricamento dati da: ', record_name, '...']);

% Prova a caricare i dati

  [signal, Fs, tm] = rdsamp(record_name);

% Calcola le componenti AP, ML, V corrette per tilt e gravità

    col_V = 1; % Indice colonna Verticale

    col_ML = 2; % Indice colonna Medio-Laterale

    col_AP = 3; % Indice colonna Antero-Posteriore

  [aAP, aML, aV] = algo_Moe_Nilssen(signal(:,col_AP), signal(:,col_ML), signal(:,col_V), 'tiltAndNoG');

% --- 2. Parametri dell'Analisi---

window_sec = 1; % Finestra di analisi per SMA ed Energia

window = Fs * window_sec;              % Campioni per finestra da 1s

min_bout_duration_sec = 60;            % Durata minima bout di cammino

microbreak_tolerance_sec = 1;          % Tolleranza per pause

threshold_SMA = 0.135;

threshold_en = 0.05;

% Filtro passa-alto

Wn = 0.5 / (Fs / 2);

order = 4;

[B_hp, A_hp] = butter(order, Wn, 'high');

% --- 3. Pre-Processing (Filtraggio) ---

disp('Filtraggio dei segnali (filtfilt)...');

    aV_filt = filtfilt(B_hp, A_hp, aV);

    aML_filt = filtfilt(B_hp, A_hp, aML);

    aAP_filt = filtfilt(B_hp, A_hp, aAP);
```

```
% --- 4. Classificazione in finestre da 1s ---
limit = floor(length(aV_filt) / window);
if limit < min_bout_duration_sec
    warning('File %s troppo corto (%d sec) per l''analisi. Interruzione.', record_name, limit);
    return;
end
SMA = zeros(limit, 1);
energy = zeros(limit, 1);
fprintf('Analisi di %d finestre da %d secondo...\n', limit, window_sec);
print_every = 5000; % Stampa un aggiornamento ogni 5000 finestre (circa 1.4 ore di dati)
for i = 0:(limit-1)
    start_idx = window*i + 1;
    end_idx = window*(i+1);
    aVw = aV_filt(start_idx:end_idx);
    aMLw = aML_filt(start_idx:end_idx);
    aAPw = aAP_filt(start_idx:end_idx);
    % Calcolo SMA
    SMA(i+1) = mean(abs(aVw) + abs(aMLw) + abs(aAPw));
    if mod(i, print_every) == 0 && i > 0
        fprintf('... Finestra %d di %d (%.1f %%)\n', i, limit, (i/limit)*100);
    end
    % Calcolo Energia (pwelch)
        nfft = 2^nextpow2(window);
        % Parametri pwelch robusti per finestre corte
        [paAPw, f] = pwelch(aAPw, hamming(floor(window/4)), [], nfft, Fs);
        paAPw_band = paAPw(f > 0.5 & f < 3);
        f_band = f(f > 0.5 & f < 3);
```

```
        if ~isempty(f_band)
            energy(i+1) = trapz(f_band, paAPw_band);
        else
            energy(i+1) = 0;
        end
end
% Stampa il completamento finale
fprintf('... Finestra %d di %d (100 %%)\n', limit, limit);
fprintf('Calcolo SMA ed Energia completato.\n');
% --- 5. Decisione cammino (Logica OR) ---
walking = (SMA > threshold_SMA) | (energy > threshold_en);
% --- 6. Pulizia microbreaks ---
disp('Pulizia micro-breaks (medfilt1)...');
med_filter_size = (microbreak_tolerance_sec * 2) + 1; % Finestra 3 per 1s
walking_cleaned = medfilt1(double(walking), med_filter_size);
% --- 7. Bout validi (>60s) ---
disp('Identificazione bout validi...');
% Trova transizioni 0->1 (start) e 1->0 (end)
d = diff([0; walking_cleaned; 0]);
bout_starts = find(d == 1);
bout_ends = find(d == -1) - 1;
bout_durations = bout_ends - bout_starts + 1; % Durata in secondi (finestre da 1s)
walking_final = zeros(limit, 1);
valid_bouts = 0;
for j = 1:length(bout_durations)
    if bout_durations(j) > min_bout_duration_sec
        walking_final(bout_starts(j):bout_ends(j)) = 1;
        valid_bouts = valid_bouts + 1;
```

```
    end
end
fprintf('Trovati %d bout di cammino > %d secondi.\n', valid_bouts, min_bout_duration_sec);
% --- 8. Percentuale cammino ---
p_walking = sum(walking_final) / length(walking_final) * 100;
fprintf('\n--- RISULTATO ---\n');
fprintf('File: %s\n', record_name);
fprintf('Durata totale analizzata: %.1f minuti (%.1f ore)\n', limit / 60, limit / 3600);
fprintf('Percentuale cammino finale (bout > %ds): %.2f %%\n', min_bout_duration_sec, p_walking);
```