Design Lab

Digital Pedometer using FRDM-KL05Z evaluation board and MATLAB for algorithm optimization

Mirosław Baca

PROJECT OBJECTIVE/PROBLEM DESCRIPTION

The primary objective of this project was to develop a real-time system that analyzes accelerometer data to detect steps—both walking and running—and to facilitate bidirectional communication between a computer (running MATLAB) and a microcontroller. To achieve this, the project implements both simple difference-based methods and more advanced local maxima detection techniques, enhanced using High-Pass (HPF) and Bandpass (BPF) filters. These filters isolate the frequency ranges corresponding to running and walking, respectively, while adaptive logic dynamically selects the most appropriate detection method based on the signal amplitude. The system transmits accelerometer data to MATLAB via UART, and incoming commands ("WALK++" and "RUN++") are used to update and display the current step counts on an LCD. This comprehensive approach ensures robust step detection in various real-world scenarios while providing clear, real-time feedback on the device's performance.

PREPARING THE EVALUATION BOARD

1. The board

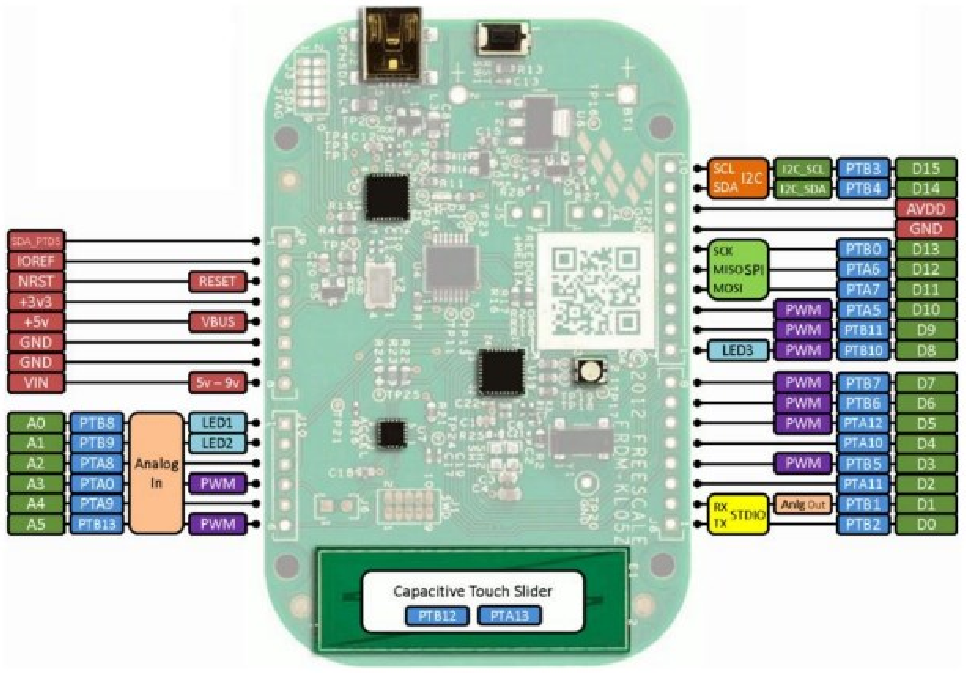

Figure 1 - FRDM-KL05Z Pin Layout

The FRDM-KL05Z evaluation board is an ideal choice for this project as it comes with a built-in accelerometer, the MMA8451, which allows for easy acquisition of 3-axis accelerometer data.

2. Keypad

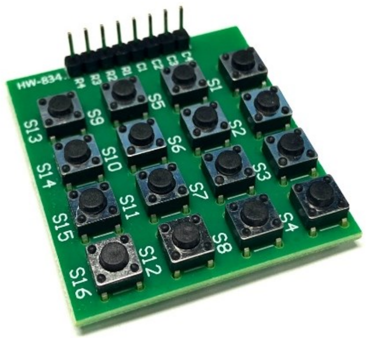
Figure 2 - 4x4 Matrix Keypad Module

The KL05Z board exposes multiple GPIO pins, including PTA11. A 4x4 matrix keypad can be attached so that each row (R1–R4) and column (C1–C4) corresponds to a KL05Z GPIO line. R4 (Row 4) of the keypad is connected to PTBA11 on the microcontroller. C4 (Column 4) of the keypad is connected to GND. The microcontroller can determine button S1 is pressed based on signal changes on PTBA11.

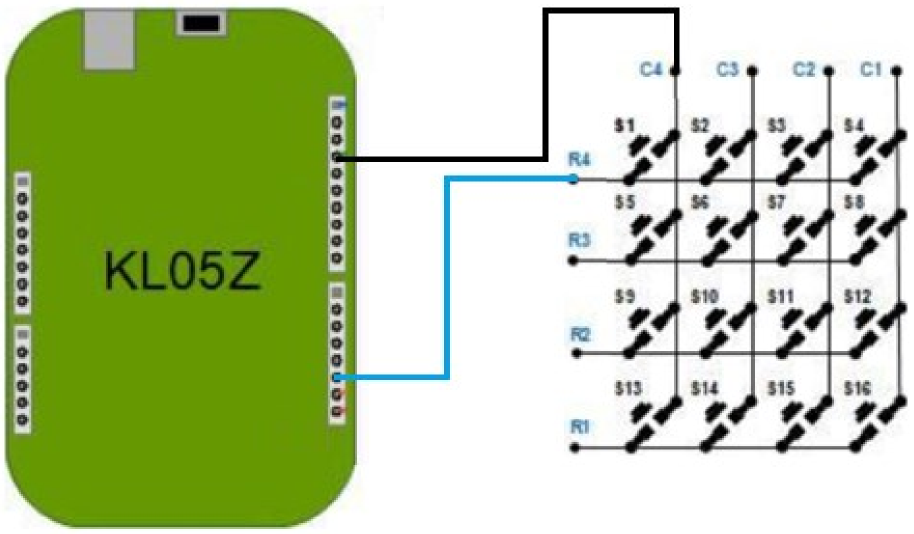

Figure 3 – Pins connection configuration

3. LCD Module

A standard 16x2 HD44780-based LCD module is used for displaying step counts and messages. Thanks to I2C expander (HD44780U) only SDA (PTB3) and SCL (PTB4) lines are connected to the LCD, along with power and ground. The PTB3 and PTB4 pins of the FRDM-KL05Z serve as the I2C SCL and SDA lines, facilitating serial data transmission to the LCD module. The VCC pin of the LCD is connected to the +5V supply from the microcontroller, while the GND pin is linked to the microcontroller's ground.

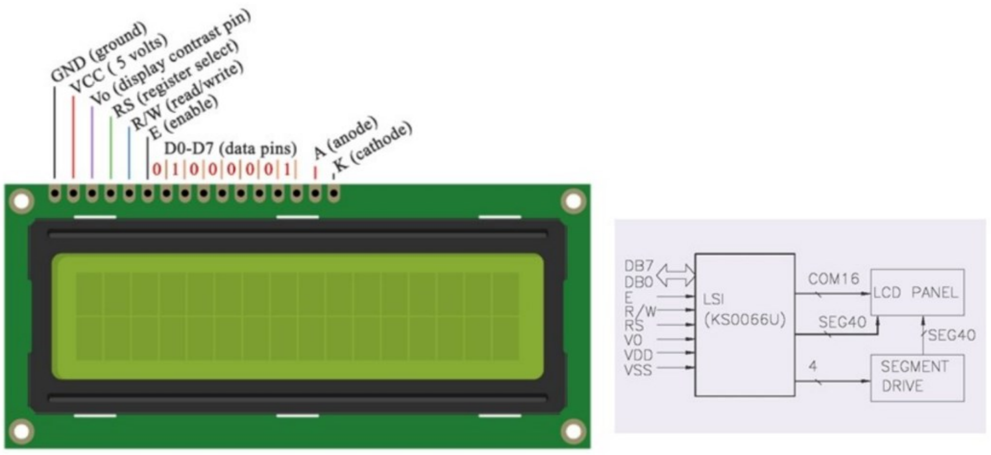

Figure 4 - LCD display module with block diagram

4. Accelerometer Interface

An on-board accelerometer (MMA8451) at address 0x1D is read via the same I2C bus as used for the LCD. Every 100ms loop continuously reads acceleration data from a sensor via the I2C interface (code below). The accelerometer's raw X, Y, and Z-axis data are retrieved, processed into a human-readable format by converting them to acceleration values in *g* units, and stored in a buffer. Finally, the formatted acceleration data is transmitted via UART to an external receiver.

```
while (true)
{
  // Accelerometer config & reading
  I2C::writeReg(0x1D, 0x2A, 1);
  I2C::writeReg(0x1D, 0x0E, 0x00);

  static char tempBuffer[36];
  static uint8_t arrayXYZ[6];
  I2C::readRegBlock(0x1D, 0x01, 6, arrayXYZ);

  double x_=((double)((int16_t)((arrayXYZ[0]<<8)|arrayXYZ[1])>>2)/4096);
  double y_=((double)((int16_t)((arrayXYZ[2]<<8)|arrayXYZ[3])>>2)/4096);
  double z_=((double)((int16_t)((arrayXYZ[4]<<8)|arrayXYZ[5])>>2)/4096);
  sprintf(tempBuffer,"%1.4f  %1.4f  %1.4f", x_, y_, z_); // default 4096
  counts/g sensitivity
  start.println(tempBuffer);


  DELAY(100);
}
```

DATA PROCESSING IN MATLAB ENVIRONMENT

1. UART connection establishment and data preprocessing

The MATLAB script first attempts to set up a serial port for communication with the microcontroller. In the testing environment, board was connected to serial port COM6 and a 9600 baud rate was chosen because it is both a common standard and matches the UART settings configured on the microcontroller (MCU). If MATLAB fails to open COM6, the script closes any open figure or serial connection and terminates, preventing further errors.

```
function matlabFilterTests()

    %% ======================= UART & Global Variables ========================
    % Configure the serial port
    serialPort = serialport('COM6', 9600);
```

Once the serial connection is established, incoming accelerometer data (X, Y, Z) is read line by line in the main reading loop. Each line is split into numeric values and filtered to discard incomplete messages (fewer than 3 values).

2. Testing parameters and filters

After many test final parameters and filters that were defined:

- **Sampling Rate** (10 Hz):
  This rate was chosen because the board uses a delay of 100 ms between samples, resulting in a 10 Hz sampling frequency. This frequency is sufficient for capturing the dynamics of walking and running. The initial frequency of 5 Hz (200 ms) was unfortunately not sufficient to measure running.
- **High-Pass Filter** (HPF) **Cutoff** (2 Hz):
  Only displays higher frequencies associated with running and get rid of the constant component (1g) and slow movements.
- **Bandpass Filter** (BPF) **Range** (0.3–2 Hz):
  Focuses on typical walking frequencies excluding constant component of the gravitational acceleration (1g).
- Thresholds for basic methods (difference-based):
  oldMethodStepThreshold = 0.2 (for walking) and **oldMethodRunThreshold** = 0.7 (for running).
- Thresholds for advanced calculation methods (local maxima):
  secondMethodThreshold = 1.05 for the raw magnitude.

- The parameters ***hpfPeakThreshold*** and ***bpfPeakThreshold*** (both set to 0.3) define the minimum amplitude required for a local maximum to be considered a valid step candidate in the high-pass and bandpass filtered signals, while ***minHPFSampleDist*** (3 samples, i.e., 0.3 s) and ***minBPFSampleDist*** (7 samples, i.e., 0.7 s) ensure that successive peaks are separated by a sufficient time interval to prevent multiple detections for a single step. The ***runWalkThreshold6*** parameter, set at 1.3, serves as a decision boundary to classify the detected movement: if the filtered signal's magnitude exceeds 1.3, it is treated as a run; otherwise, it is considered a walk.

```
 % Thresholds and minimal sample distance for local maxima

hpfPeakThreshold = 0.3;
bpfPeakThreshold = 0.3;
minHPFSampleDist = 3;  % at Fs=10 -> 0.3 s
minBPFSampleDist = 7;

% Threshold for run/walk decision on Subplot 6
runWalkThreshold6 = 1.3;
```

3. Choosing the right algorithm preprocessing

The measurements described in this section can be seen in the graphic on the next page.

Initially, the MATLAB code was configured to simply display the raw X, Y, and Z accelerometer axes without any markers or filters to verify that the data acquisition was working correctly. Once the basic data integrity was confirmed, a simple difference-based algorithm was implemented to mark steps by detecting changes between consecutive samples. However, this approach produced many false detections—especially during early tests when the sampling frequency was set to 5 Hz instead of the optimal 10 Hz.

To improve accuracy, the algorithm was enhanced by incorporating local maximum detection. In this approach, a small sliding window was maintained to identify peaks that stood out compared to their immediate neighbors, thus reducing false positives. Nevertheless, a challenge remained: the accelerometer naturally registers a value of approximately 1 g even when stationary due to gravity. This constant offset led to unintended detections, as movements could be misinterpreted.

To address this, a high-pass filter (HPF) was introduced, with its cutoff frequency carefully chosen to remove both the constant gravitational acceleration and the slower motions associated with walking. This adjustment left primarily the rapid movements indicative of running. However, since normal walking steps were also required to be counted, a bandpass filter (BPF) was then added as a second plot, which specifically targeted the frequency range typical for walking.

Ultimately, the detection logic was split: the HPF was used for running detection (where its higher sensitivity to rapid changes was beneficial), while the BPF was used to accurately count slower walking steps. This dual-filter approach is demonstrated on the final sixth subplot, where markers from both filters are shown—each labeled accordingly in the legend. This combined and optimized algorithm effectively distinguishes between walking and running, ensuring that each type of movement is accurately captured despite the inherent challenges of accelerometer readings.

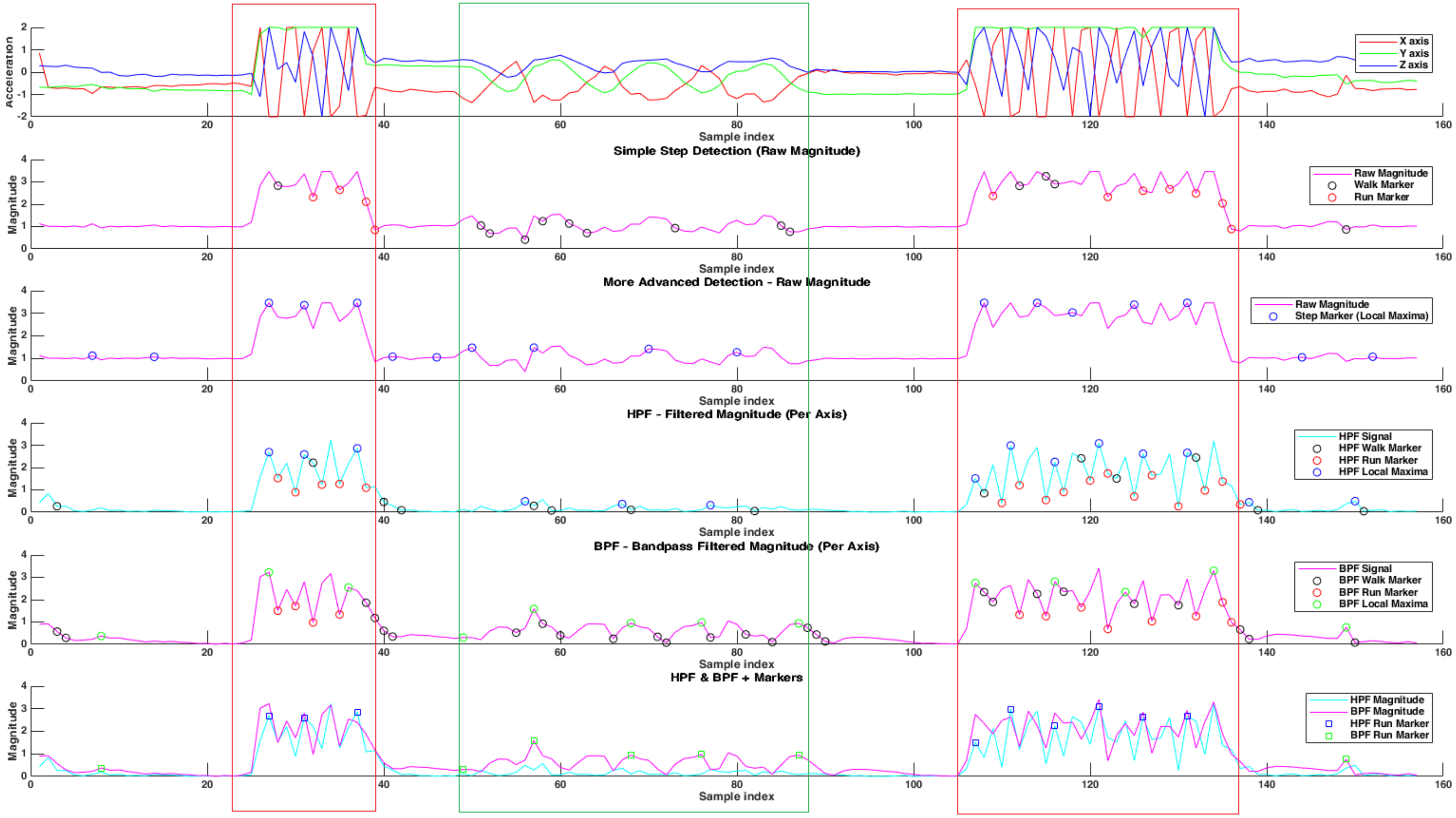

Figure 5 - Graphs showing the development of the algorithm for measuring steps. In the red rectangle we can observe simulations of running, in the green rectangle we can observe simulations of walking. As could be seen, the best accuracy of step recognition was obtained in the last graph using both filters, such a solution, although it is far from ideal, also mostly eliminates the recognition of noises as steps.

4. Final code and sending back to UART

After many tests an optimal solution have been implemented. Final script after computing step detections, send step-related messages ("WALK++" or "RUN++") back to the microcontroller via UART. A serial port remains open, enabling MATLAB to write strings (using writeline(serialPort, 'WALK++')). The microcontroller, in turn, can interpret these messages to update LCD displays and counters.

CODE STRUCTURE IMPLEMENTED ON THE FRDM-KL05ZJ BOARD

1. File structure and programming environment

The project was developed using Keil uVision, and the manufacturer's libraries that abstract direct register manipulation. Despite these conveniences, the implementation required manual creation of seven source/header files for complete functionality like Uart, I2C communication, interrupt handling etc. The file structure is as follows:

1.1. main.cpp

Serves as the entry point for the FRDM-KL05Z firmware. It configures the system peripherals, sets up UART communication on COM6 (baud rate 9600), initializes the LCD via I^{2}C, and handles a button interrupt (on PORTA) to reset step counters. The main loop for reading and transmitting data from the accelerometer is also here.

1.2. Uart.cpp/Uart.hpp

Implements the UART communication interface, including initialization, data transmission, and interrupt-driven reception. The UART interrupt handler processes incoming messages and updates the step counters accordingly.

1.3. BoardSupport.cpp/BoardSupport.hpp

Contains low-level board support functions for LED, I^{2}C. These modules abstract hardware operations, such as managing I^{2}C communications, thereby simplifying higher-level application development.

1.4. Lcd.cpp/Lcd.hpp

Implements the driver for a LCD using the I^{2}C expander. The code handles initialization, cursor positioning, and display functions (including backlight and blink control) without requiring manual I^{2}C transaction coding.

2. Encountered problems and optimizations

Initially, the sampling frequency was limited to 5 Hz due to a 200 ms delay, which resulted in unreliable data processing. After optimizing the code it became safe to set the delay to 100 ms, achieving a stable 10 Hz sampling rate, which significantly improved real-time performance. These optimizations ensured smooth operation even under increased processing loads.

3. Interrupt handling

The firmware uses two key interrupts:

- **Button Interrupt (PORTA_IRQHandler)**:
  Triggered by a falling edge on the designated button pin (PTA11). The ISR clears the interrupt flag, resets the step counters (WalkStep and RunStep), and updates the LCD to inform the user that the step counts have been reset.

- **UART Interrupt (UART0_IRQHandler)**:
  Activated upon receiving data via UART. The ISR collects incoming characters into a buffer until a newline or carriage return is detected. Based on the received command (e.g., "WALK++" or "RUN++"), it increments the appropriate step counter and updates the LCD with the current counts.

Together, these components enable a robust, real-time system where the microcontroller reads sensor data, communicates with MATLAB via UART on COM6 (using the standard 9600 baud rate), and provides immediate feedback through the LCD display.

PROJECT SUMMARY AND CONCLUSIONS

This project implements a real-time step detection system using an FRDM-KL05Z microcontroller and MATLAB. Accelerometer data is continuously transmitted via UART (COM6, 9600 baud) and processed in MATLAB, where a series of detection strategies were applied. The development process evolved from simple difference-based detection to advanced techniques incorporating local maxima and filtering (HPF and BPF). Main optimizations that have been achieved:

- Filters to isolate frequency ranges,
- Minimum distances to prevent duplicate peaks,
- Adaptive thresholds to decide whether running or walking

On the microcontroller side, the firmware is organized into multiple modules (main, UART, BoardSupport, LCD). The system utilizes interrupt-driven mechanisms—one for button presses to reset step counts and another for UART data reception—to create a real-time feedback loop. This loop, which involves receiving accelerometer data, detecting steps, and returning messages to MATLAB, ensures that the system can be further optimized and adapted for other devices beyond the FRDM-KL05Z.

Main features of the code on the MCU:

- **Developed in C++:**
  Uses the object-oriented features of C++ in Keil uVision, enhancing code structure and reusability.
- **High Performance:**
  Optimized delays and efficient execution ensure real-time operation even with intensive sensor data processing.
- **Interrupt-Driven Design:**
  Implements both button and UART interrupts for immediate and responsive handling of external events and data reception.
- **Modular Structure:**
  Code is divided into clearly separated modules (main.cpp, Uart.cpp/Uart.hpp, BoardSupport.cpp/BoardSupport.hpp, Lcd.cpp/Lcd.hpp), making it highly readable and maintainable.
- **Scalability:**
  The modular architecture and use of manufacturer libraries facilitate easy adaptation to other devices or additional peripherals.
- **Efficient Resource Management:**
  Leverages low-level hardware abstraction while minimizing the need for manual register manipulation, thus reducing development complexity and overhead.

Overall, the integration of robust filtering, adaptive detection, and a modular code structure has resulted in a highly efficient and reliable step detection solution.