UrPersonalTrainer Project Report

Yuhang Zhang
Faculty of Engineering
McGill University
Montréal, Canada
yuhang.zhang@mail.mcgill.ca

Rintaro Nomura
Faculty of Engineering
McGill University
Montréal, Canada
rintaro.nomura@mail.mcgill.ca

Chang Zhou
Faculty of Engineering
McGill University
Montréal, Canada
chang.zhou2@mail.mcgill.ca

I. Introduction

UrPersonalTrainer(UPT) is a wearable device that tracks trainee's squat exercises. During the training, the trainee shall attach the UPT on his/her back, and the UPT is capable of counting the number of push-ups the trainee does. Once the trainee achieves his/her daily push-ups objectives, the UTP would notify its user through a speaker and LED and, in addition, generate an exercise report indicating how many calories were burnt during the training.

II. Project Overview

A. System Block Digram

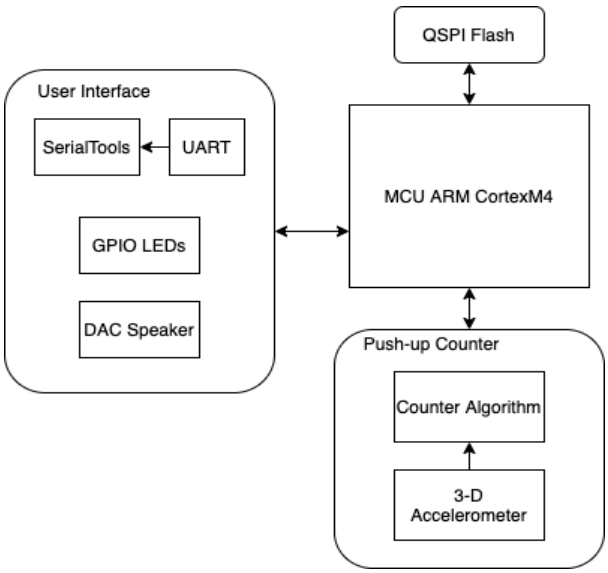

Fig 2.1. System Block Diagram.

B. Design Description

In this project, we require an STM32L475 MCU discovery board with a built-in accelerator sensor and a gyroscope sensor, a USB 2.0 A male to Micro B cable, a speaker, and a computer for development.

The UPT software system consists of four main components: the MCU, the User Interface (UI), the Push-up counter, and the QSPI Flash respectively, as shown in the Fig 2.1.

The Push-up counter reads acceleration information from the built-in 3-D accelerometer. This system shall sample the accelerometer readings every 0.1 seconds so that the digital reading could adequately represent the overall system dynamics. In addition, the Counter Algorithm would interpret the accelerometer reading and inform the MCU of the number of push-ups that the UPT user has done. The MCU would keep track of the number of push-ups and ask the UI to output training information once the training objective is approached. The QSPI Flash is also essential for our system because the DAC is intended to play a variety of sounds and may require a relatively large memory space to pre-store these output sound simulations. Furthermore, once the MCU triggers the output, the UI generates DAC and GPIO output via the speaker and LED respectively, and informs the user verbally and virtually that today's push-up target has been achieved. Fig. 2.2. shows how the UPT should be used.

Fig 2.2. Demonstration of using the UPT.

III. Design Details

A. Design of the Push-up Counter

The Push-up Counter reads the accelerometer result and counts the number of push-ups. We consider the 3-D accelerometer is the designated sensor capable of understanding and interrupting the user's push-up dynamics. After configuring the above I2C sensor, we began by discovering the output behavior of the accelerometer readings. The output of the accelerometer is an array containing three independent variables: the acceleration on the x-axis, the y-axis, and the z-axis respectively. We consider the acceleration reading on the z-axis, a.k.a. the vertical acceleration, would be the parameter we require for the push-ups counter.

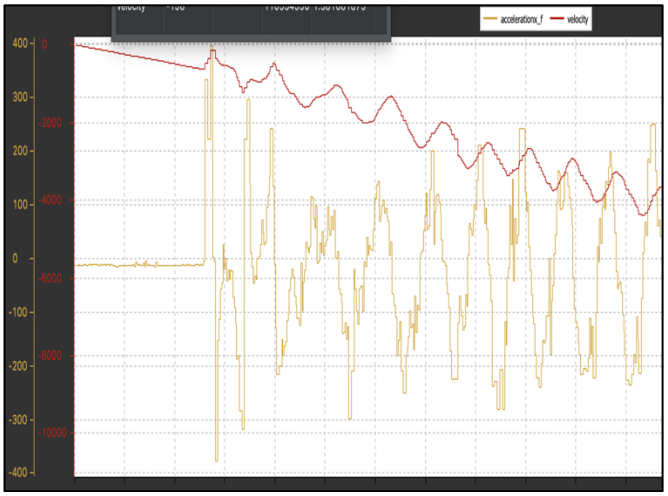

Fig 3.1. Unfiltered Z-axis Accelerometer Reading (Orange) and Unfiltered Integrated Signal (Red).

As shown in Fig 3.1., the orange signal indicates the raw z-axis accelerometer reading when the mcu was simulating the push-up movement. We can observe from the figure that the raw accelerometer reading experiences a major oscillation whenever one push-up is performed. However, by testing this accelerometer reading, we found that the sensor readings were sometimes rather noisy.

Ideally, we want to estimate the underlying signal without noise, introducing as little distortion as possible. Therefore, we consider it would be a good practice to apply an extreme number filter for the sampled data. We started by designing a low-pass/high-pass filter to reduce the input data noise. The filter works in the way that it eliminates extreme signals by removing everything outside the sampling filter range which, in our project, is considered as undesired aspects from the measurement.

Another noise elimination method implemented was the measurement integration. This idea came for the PID control problem. Integration trends to eliminate random noises [1], and the high-frequency noise is normally considered to be random and additive to a measured signal and is usually uncorrelated in time. Practically in our project, we were using the Trapezoidal rule to approximate the signal integration.

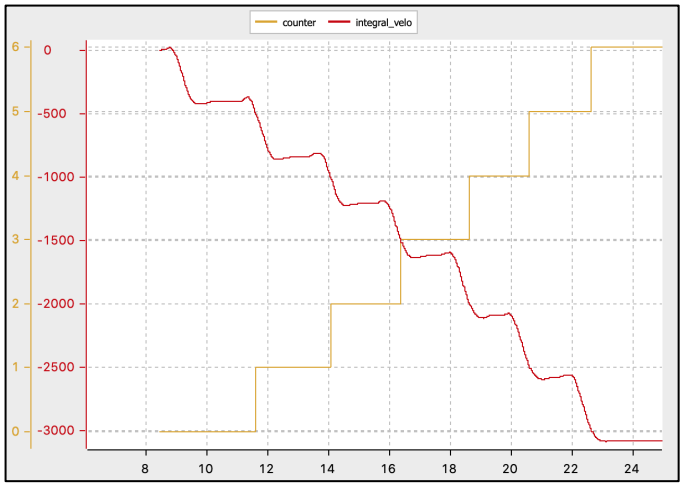

Fig 3.2. Filtered Integrated Signal (Red) and Counter Result (Orange).

For the software implementation, we constructed an algorithm that would keep integrating filtered measurement samples using the Trapezoidal method. As we could see from the above figure, the filtered integrated measurement signal illustrates a smooth, distinguishable output that can be clearly correlated to the UPT dynamic movement.

To interpret the filtered integrated signal, we established a push-up counter algorithm shown in Fig 3.3. constructed an integer array arr recording the integrated acceleration measurement at each push-up. Whenever the integrated measurement experiences a major decrement, we shall increase the push-up counter by one and update the array for the latest push-up measurement. From Fig 3.2, we would see the correlation between the counter result and the filtered integrated signal.

```
696                 if(abs(integral_velo/100 - arr[counter])> 4){
697                         counter++;
698                         arr[counter] = integral_velo/100;
699
700                         if(counter == target){
701                                 beepMany();
702                         }else{
703                                 beepOnce();
704                         }
705
706                 }
```

Fig 3.3. Push-up Counter Algorithm

In addition, the algorithm shall compare the number of push-ups completed with the daily target and trigger the corresponding user interface functions. Function *beepMany()* is triggered when the daily training target is approached. It would notify the trainee verbally and virtually: the speaker will play a 3-second piece of music, and the LED is turned on indicating the trainee his/her target has been accomplished. The *beepOnce()* function will only play a note and will be called every time when one push-up is finished.

B. Implementation of the User Interface

Once a push-up has been completed by the UPT user, the configured LED shall bink once and the speaker shall play a short note. In addition, if the user's daily push-up objective was achieved, the UI shall also inform the user verbally and visually and play a 2-second piece of music. In order to do so, we have completed the setup and initialization of the DAC and GPIO pins.

- UART: We have implemented the UART to send and output the accumulated push-up numbers whenever the push-up has been detected. Once we have completed the push-up counter component, the output text will be in a little more detail: for example, including the burnt calories from achieving the goal. Once the goal has been achieved, the personal training report will be sent through UART, which includes all the push-up data points and total number of push-ups. This should be useful because the user may use other software, e.g., excel, to generate a graph.
- GPIO LEDs: We are using LEDs as an indicator that the push-up has been detected. For this part, we are simply blinking the LED when the push-up has been detected.
- DAC Speaker: We are using a beep to notify the user when the goal has been achieved. To do so, we are creating a few sound waves while the system is starting up, and sending it to the DAC once the goal has been achieved.

C. Implementation of the Calorie Calcualtor

To generate an exercise report that is capable of informing the trainee how much celeries were brunt during the push-up straining, we shall utilize the following formula.

$$C = \frac{M_{Trainee} \times F_g \times L_{arm}}{4148} \qquad (1)$$

- C is the burnt calorie.
- $M_{Trainee}$ is the net mass of the trainee.
- F_g is the gravitational force.
- L_{arm} is the trainee's arm length in meter.
- 4184 is the unit constant converting work spent from Joules into Calories.

D. Implementation of the QSPI Flash

The DAC speaker intends to play some simple sounds. However, it still requires a relatively large memory to prepare these output sounds. Therefore, we are using QSPI Flash to store the sound waves to be used in DAC. We do not expect to require an entire Flash memory space for this. Therefore, in this project, we have been using 2 flash memory blocks that are efficient for our short piece of music.

IV. Testing

In this project, various tests are deployed to ensure the project would work as desired. In this section, we construct unit tests for each subsystem and integration for the whole project.

A. Push-up Counter Testing

Throughout testing, we utilized the USART and SWV to observe the behaviors of the unfiltered z-axis acceleration, filtered z-axis acceleration, unfiltered integrated measurement, and the filtered integrated measurement. We construct a push-up counter unit test which generates a simulated push-up dynamic that informs the tester that if the current system has the competence to counter the number of push-ups. With the unit test, rather than perform an actual push-up, we could test the counter algorithm by merely running the unit test. Therefore, we could test the algorithm efficiently and effectively. In addition, we can overview the currently deployed system and understand which part of the counter algorithm should be modified.

B. User Interface Testing

The user interface system in this project includes one LED, one speaker, and a UART protocol. What we did to test these components was to create individual unit test functions. Unit tests call simulation functions which trigger the above components, and we shall see if they are working as desired.

C. QSPI Flash Testing

Throughout the project, we have tested a few alternative ways to process the signals from the sensors. As mentioned in the initial report, the first approach we took is to take a double integral of the acceleration value to obtain the distance, which could be compared with the arm length to obtain the number of push-ups. However, due to the high level of noise from the accelerometer itself and the human body motion, it was premature to determine the precise distance through our signal processing implementation. For a similar reason, we were also unsuccessful to detect the push-up motion by observing the acceleration motion by itself.

V. Parameter Configuration

Throughout the project, multiple parameters had to be set to achieve the prescribed behaviors in the project definition.

A. Z-direction Acceleration Sampling Threshold

One of the most significant parameters in the design process is the z-direction acceleration threshold. In our attempt, a measurable push-up can be recognized and counted if the accelerometer reading in the vertical direction, z, exceeds the z-direction acceleration threshold. What we did to configure this specific parameter is testing the threshold many times and summarizing a reliable reading. We modified the code in the way that the accelerator sensor would, via USART, generate its z-direction sensor reading while doing push-ups. During the testing, we attached the MCU to our back and performed push-ups. By observing those readings, it was obvious that the z-direction accelerometer sensor reading experienced two plateaus at each push-up. To keep the

threshold reliable, we tested 20 times and chose their minimum value as the z-direction acceleration threshold.

B. Low-Pass/High-Pass Sampling Filter Range

To implement the low-pass/high-pass filter, we need to define a sampling range where signals situated inside are preserved, whereas anything outside is considered as noise extremes and will be ignored. We configured the sampling range by taking measurements of one push-up ten times and took averages of upper amplitude boundary and lower amplitude boundary respectively. Therefore, the upper boundary average and the lower boundary average together define the filter range.

VI. Result and Evaluation

To evaluate our UPT project, we firstly simulated the push-up dynamics by waving the MCU up and down, and in the meantime, checked if our UPT can successfully distinguish and recognize a push-up movement and make correct responses. We, therefore, settled the board on our shoulders and performed push-ups. After tweaking some minor parameters, we found the MCU was still capable of counting the number of push-ups. In addition, we examined the output features and the specification described in the proposal to ensure every requirement has been satisfied. In the end, we consider our design can correctly detect the movement and generate expected responses, which hence meet the project requirement.

VII. Further Improvement

Future improvements and recommendations of the UPT project can be summarized into these three following aspects:

1) Include a timer to keep track of the exercise time of the user. By including a timer, we can provide a more comprehensive exercise report with the user's exercise speed and we can also add a record to keep track of the shortest time the user took to complete 100 push-ups.

2) Develop different exercise modes such as running mode and abdominal curl mode to enrich the functionalities of our UPT project. For the running mode, we could measure the speed of the user and the distance completed by the user; we could then make use of those data together with the exercise time, the weight, and the age of the user to estimate calories burnt by the user. For the abdominal curl mode, we could measure the number of abdominal curls done by the user in a similar approach as we measure the number of push-ups done by users and estimate the calories burnt by the user.

3) Implement a Wi-Fi connection. The X-Cube-WiFi1 expansion software package for our project runs on STM32 and could be used for building Wi-Fi applications with the SPWF01SA module; by doing this, we can free our development board from USB cable to the computer while UART transmission.

References

[1] Cesare V. Parise ,Marc O. Ernst, “Noise, multisensory integration, and previous response in perceptual disambiguation” PLoS Comput Biol 13(7): e1005546. *(references)*.