REPORT

전자공학도의 윤리 강령 (IEEE Code of Ethics)

(출처: http://www.ieee.org)

나는 전자공학도로서, 전자공학이 전 세계 인류의 삶에 끼치는 심대한 영향을 인식하여 우리의 직업, 동료와 사회에 대한 나의 의무를 짐에 있어 최고의 윤리적, 전문적 행위를 수행할 것을 다짐하면서, 다음에 동의한다.

1. **공중의 안전, 건강 복리에 대한 책임**: 공중의 안전, 건강, 복리에 부합하는 결정을 할 책임을 질 것이며, 공중 또는 환경을 위협할 수 있는 요인을 신속히 공개한다.
2. **지위 남용 배제**: 실존하거나 예기되는 이해 상충을 가능한 한 피하며, 실제로 이해가 상충할 때에는 이를 이해 관련 당사자에게 알린다. (이해 상충: conflicts of interest, 공적인 지위를 사적 이익에 남용할 가능성)
3. **정직성**: 청구 또는 견적을 함에 있어 입수 가능한 자료에 근거하여 정직하고 현실적으로 한다.
4. **뇌물 수수 금지**: 어떠한 형태의 뇌물도 거절한다.
5. **기술의 영향력 이해**: 기술과 기술의 적절한 응용 및 잠재적 영향에 대한 이해를 높인다.
6. **자기계발 및 책무성**: 기술적 능력을 유지, 증진하며, 훈련 또는 경험을 통하여 자격이 있는 경우이거나 관련 한계를 전부 밝힌 뒤에만 타인을 위한 기술 업무를 수행한다.
7. **엔지니어로서의 자세**: 기술상의 업무에 대한 솔직한 비평을 구하고, 수용하고, 제공하며, 오류를 인정하고 수정하며, 타인의 기여를 적절히 인정한다.
8. **차별 안하기**: 인종, 종교, 성별, 장애, 연령, 출신국 등의 요인에 관계없이 모든 사람을 공평하게 대한다.
9. **도덕성**: 허위 또는 악의적인 행위로 타인, 타인의 재산, 명예, 또는 취업에 해를 끼치지 않는다.
10. **동료애**: 동료와 협력자가 전문분야에서 발전하도록 도우며, 이 윤리 헌장을 준수하도록 지원한다.

위 IEEE 윤리헌장 정신에 입각하여 report를 작성하였음을 서약합니다.

<Project-3 창의 실험과제>

학 부: 전자공학과

제출일: 2021. 05. 22

과목명: 신호 및 시스템

교수명: 조위덕 교수님

분 반: C063-1

학 번: 201821094

성 명: 안재혁

1. 프로젝트 설명

• 걸음 수 검출 알고리즘

: 웨어러블 디바이스 환경에서 3축 가속도계 센서의 x, y, z축 가속도 신호를 통해 사용자의 걸음 수를 계산하는 알고리즘

• 알고리즘 동작 순서

1. G단위 신호 스케일링(단위 정량화)
2. 가속도 모멘텀 산출
3. 모멘텀 신호 전처리(노이즈 제거)
4. 모멘텀 신호 양자화
5. 피크/밸리(Peak/Valley) 검출
6. 걸음 수 검출

=> 알고리즘은 1번부터 6번까지 순서대로 진행되며, 이를 블록 다이아그램으로 나타내면 아래와 같다.

2. 단계별 알고리즘 설명 및 결과 분석

[1] G단위 신호 스케일링(단위 정량화)

① 이론 설명

가속도 센서에서 임베디드 설정에 따라 측정값의 범위가 달라질 수 있기 때문에, 가속도 크기의 단위 중 하나인 중력 단위 가속도로 변환하여 사용한다. 수식은 아래와 같다.

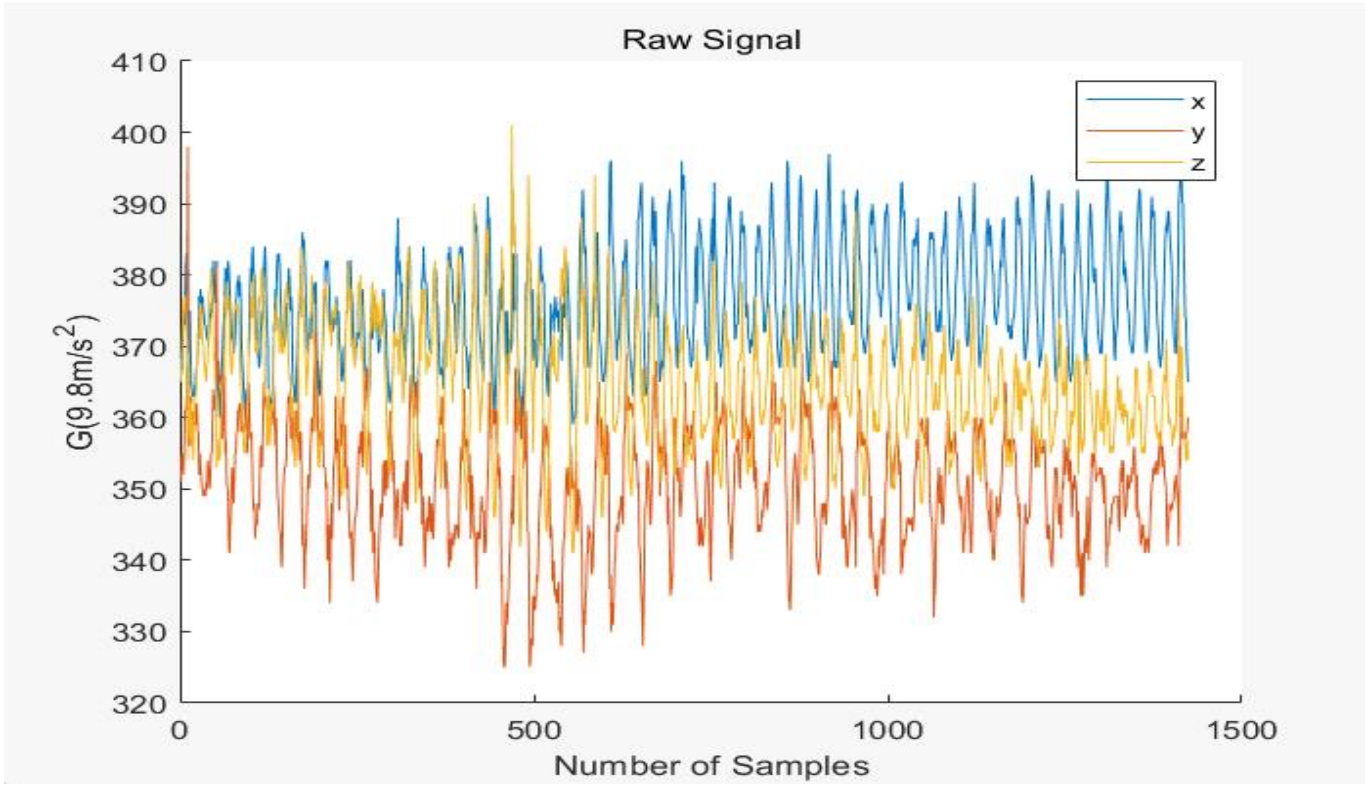
$$G_{divisor} = \frac{2^{(N_{resolution}+1)}}{G_{saturation}}$$

- $G_{saturation}$: 가속도 센서 측정 범위 설정 값, 단위는 [G]이다.
- $N_{resolution}$: ADC(Analog to Digital Converter)의 분해능 설정 값이다. 이 값은 측정하는 기기의 ADC 설정에 따라 달라지며, 이번 프로젝트에서 측정한 기기의 ADC 범위는 0부터 1023까지의 범위이므로 10(bit)을 사용하였다.
- $G_{divisor}$: 위 수식에 의해 측정할 수 있는 중력 단위 가속도 크기이다.

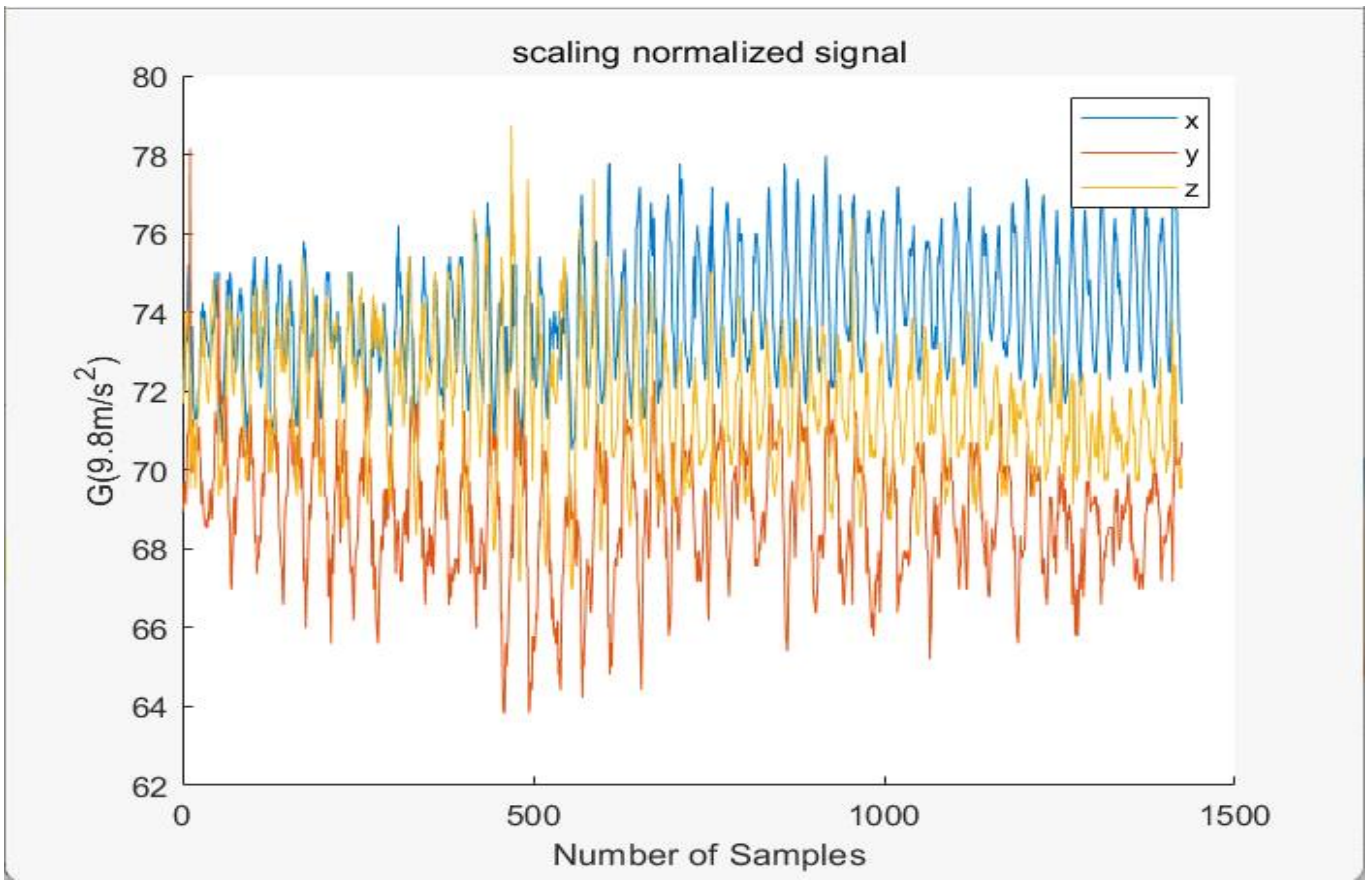
② 코드

```
%% G단위 Scaling

g = 9.81; % G = 9.80665m/s^2
G_saturation = 41 * g; % G_saturation 범위 임의 설정
N_resolution = 10; % 양자화 범위 2^10-1 까지 표현
G_divisor = 2^(N_resolution + 1) / G_saturation; % G_divisor 계산
ac = (data./G_divisor); % data 값을 G_divisor로 나눈 데이터 ac에 저장
acc = (data./G_divisor)./100;
% overflow 방지하기 위해 100으로 나눠서 데이터 acc에 저장

figure()
hold on; grid on; box on;
plot(data); % data 그래프 플롯
xlabel('Number of Samples') % x축 지정
ylabel('G(9.8m/s^2)') % y축 지정
title('Raw Signal') % 제목 지정
legend('x','y','z') % 범례 지정
hold off; grid off; box off;

figure()
hold on; grid on; box on;
plot(ac); % ac 그래프 플롯
xlabel('Number of Samples') % x축 지정
ylabel('G(9.8m/s^2)') % y축 지정
title('scaling normalized signal') % 제목 지정
legend('x','y','z') % 범례 지정
hold off; grid off; box off;

figure()
hold on; grid on; box on;
plot(acc); % acc 그래프 플롯
xlabel('Number of Samples') % x축 지정
ylabel('G(9.8m/s^2)') % y축 지정
title('overflow prevention signal ') % 제목 지정
legend('x','y','z') % 범례 지정
hold off; grid off; box off;
```

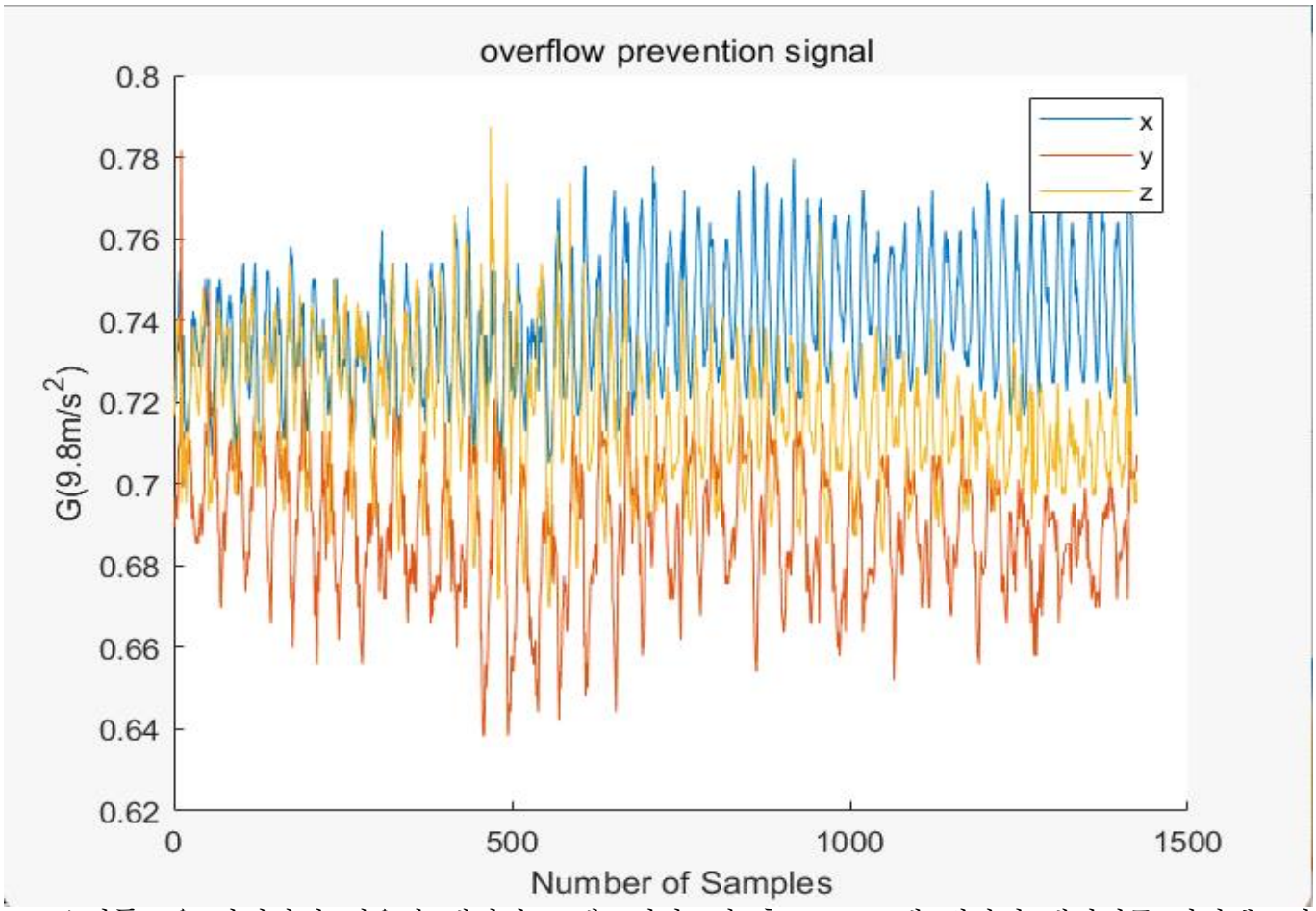
③ 그래프



=> 가속도의 raw 데이터를 plot한 그래프이다. x축은 data의 개수이며, y축은 데이터를 아직 중력 단위로 scaling 하지 않았으므로 데이터의 크기 그대로 반영되었다.



=> 가속도의 raw 데이터를 중력 단위로 scaling 하여 plot한 그래프이다. 데이터를 중력 단위 가속도 크기인 $G_{divisor}$로 나누었기 때문에 y축의 범위가 전 그래프와 달라진 것을 알 수 있다.



=> 오버플로우 방지까지 적용한 데이터 그래프이다. 각 축 x, y, z에 대하여 데이터를 나타냈으며 x축은 sample의 수이고 y축은 중력가속도 G이다. 이때 y축의 크기는 데이터를 $G_{divisor}$와 100으로 나눈 값을 적용하여 0.62부터 0.8까지 스케일을 정의하였다.

[2] 가속도 모멘텀 산출

① 이론 설명

모멘텀은 운동량으로 물체의 질량과 속도의 곱으로 표현할 수 있다. 가속도 신호에서의 모멘텀은 각 축별(x축, y축, z축) 가속도 신호에서 방향성을 제거한 가속도 크기를 뜻한다. 가속도 신호는 중력가속도를 측정하기 때문에 착용방향에 따라 다른 파형의 신호가 얻어질 수 있다. 가속도 신호의 모멘텀을 산출하면 센서의 방향성을 고려하지 않고 가속도 신호를 처리할 수 있다. 모멘텀의 수식은 아래와 같다.

$$m = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

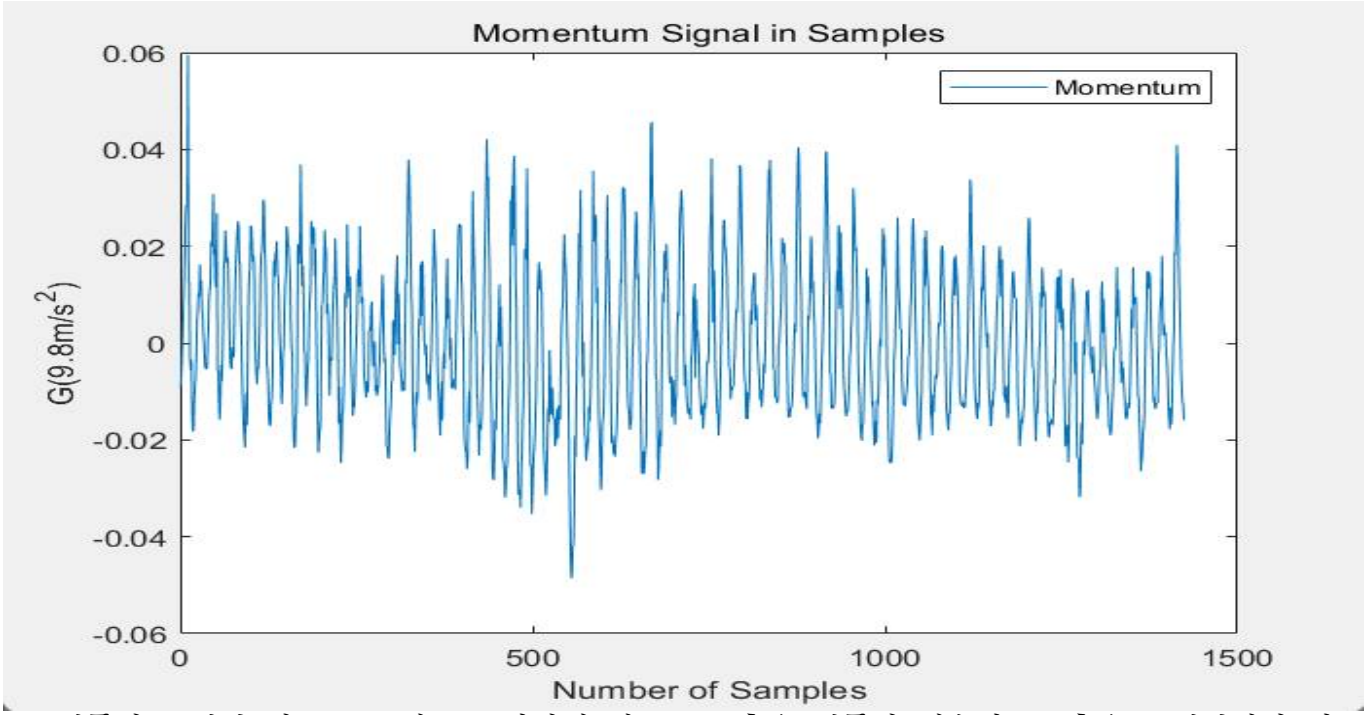
x, y, z축 속성을 이용하여 모멘텀을 구하는 수식이다. 이를 이용하여 모멘텀을 구한 후 모멘텀의

값을 0을 기준으로 normalization(표준화) 시키기 위해 평균값을 구한다. 따라서 모멘텀에서 평균값을 뺀 값을 사용한다.

② 코드

```
47      %% 가속도 모멘텀 산출
48
49      for i = 1:L
50      momentum(i) = sqrt(acc(i,1).^2 + acc(i,2).^2 + acc(i,3).^2);
51      % momentum 산출, acc(i,1) = x축 데이터, acc(i,2) = y축 데이터, acc(i,3) = z축 데이터
52      end
53      m = mean(momentum); % 모멘텀의 평균값
54      result = momentum - m; % 결과 = 모멘텀 - 평균값
55
56      figure()
57      plot(result) % 모멘텀 결과값 그래프 플롯
58      legend('Momentum') % 범례 지정
59      xlabel('Number of Samples') % x축 지정
60      ylabel('G(9.8m/s^2)') % y축 지정
61      title('Momentum Signal in Samples') % 제목 지정
62
```

③ 그래프



=> 샘플의 모멘텀 신호를 그래프로 나타낸 것으로, x축은 샘플의 개수이고 y축은 모멘텀에서 평균값을 뺀 값이다. 이때 모멘텀의 값이 0을 기준으로 위아래로 분포되어 있는 것을 볼 수 있는데, 수식으로 구한 모멘텀 값에서 평균값을 빼어 표준화(Normalization)를 진행하였기 때문이다.

=> 또한 G단위 scaling처럼 값을 x, y, z축으로 나누어 나타내지 않았고 한 종류의 값에 대하여 plot 되었다. 이는 모멘텀이 x, y, z의 방향성을 제외하여 구한 크기라는 부분에서 이유를 알 수 있다. 벡터 성분을 가지지 않고 오로지 스칼라 성분만을 지니기 때문에 위의 그래프와 같이 하나의 값으로만 나타내는 것이다.

=> 그래프로 나타난 선을 관찰하면 상당히 매끄럽지 않고 노이즈가 많이 발생했음을 알 수 있다. 진자운동처럼 걷는 것이 아니므로 노이즈가 발생하며, 이후에 이를 이용한 다른 값을 구할 때 상당한 오차가 발생할 수 있는 원인이 된다. 따라서 노이즈는 최대한 제거해주어야 하며, 이를 제거하기 위해선 데이터 전처리 과정을 진행해야 한다.

=> 위의 수식으로 구한 모멘텀의 평균값은 m = 1.2394이다.

[3] 모멘텀 신호 전처리(노이즈 제거)

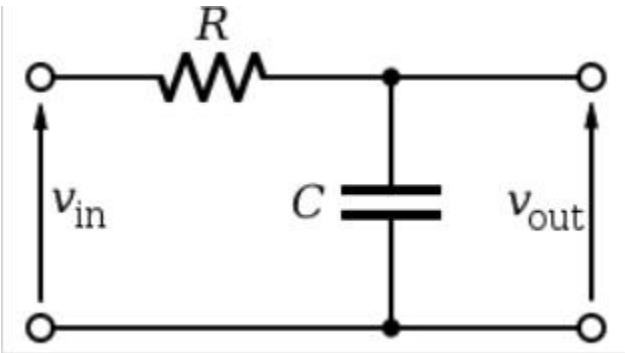
① 이론 설명

모멘텀 신호 그래프에서 노이즈를 제거하기 위해 필터를 사용한다. 이때 사용하는 필터는 Low Pass Filter이다. Matlab에서 이를 표현하기 위해 designfilt를 사용했으며 파라미터는 Sampling Frequency = 32Hz, Order = 5, Cut off Frequency = 0.5Hz이다.

• Low Pass Filter

-저주파를 통과시키는 필터를 말하며 차단주파수(Cut off frequency)보다 낮은 주파수 성분만을 통과시키는 특징이 있다. 이번 프로젝트에서 차단주파수는 0.5Hz이므로 0.5Hz보다 낮은 주파수만 통과시킨다고 할 수 있다. 주로 신호의 고주파 노이즈를 제거하기 위해 사용된다.

-Low Pass Filter의 회로 구성은 아래 그림과 같다(RC 회로)



키르히호프의 법칙을 이용하여 미분방정식을 나타내면 아래와 같고,

$$v_{\text{out}}(t) = v_{\text{in}}(t) - RC\frac{\mathrm{d}\,v_{\text{out}}}{\mathrm{d}\,t}$$

$V_{in}(t)$을 step function의 크기 V_i로 정의하여 아래 식을 해결하면

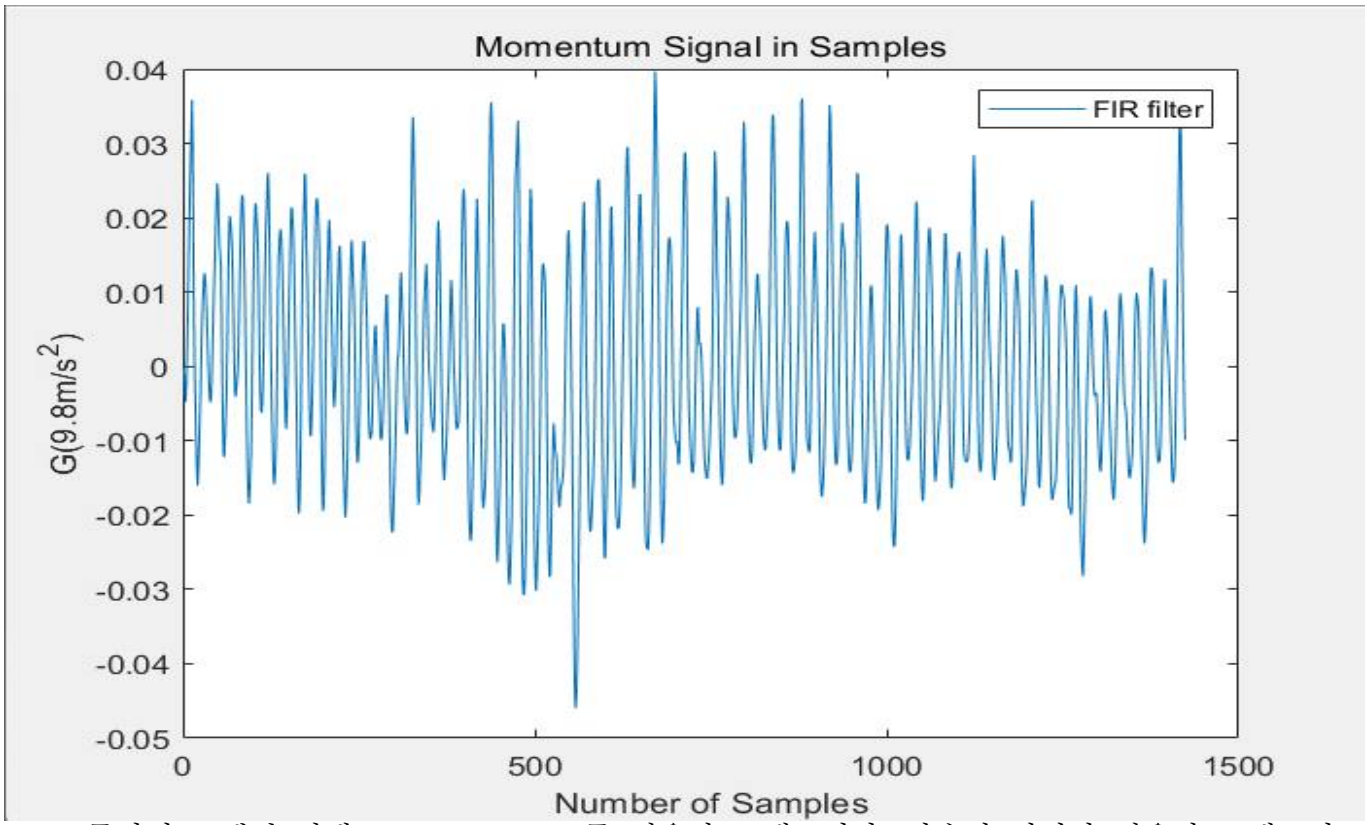
$$v_{\text{out}}(t) = V_i(1 - e^{-\omega_0 t})$$

LPF의 cut off frequency는 $w_0 = \frac{1}{RC}$임을 알 수 있다.

② 코드

```
%% Low pass filter를 통한 노이즈

Sf = 32; % Sampling Frequency = 32Hz
n = 5; % Order = 5
cf = 0.5; % Cut-off Frequency = 0.5Hz

LPF = designfilt('lowpassfir', 'FilterOrder', n, 'CutoffFrequency', cf,'SampleRate', Sf);
% Low Pass Filter 적용

momentum_filter = filter(LPF,result); % LPF를 적용한 모멘텀 신호
figure()
plot(momentum_filter) % 노이즈 제거된 모멘텀 그래프 플롯
legend('FIR filter') % 범례 지정
xlabel('Number of Samples') % x축 지정
ylabel('G(9.8m/s^2)') % y축 지정
title('Momentum Signal in Samples') % 제목 지정

```

③ 그래프



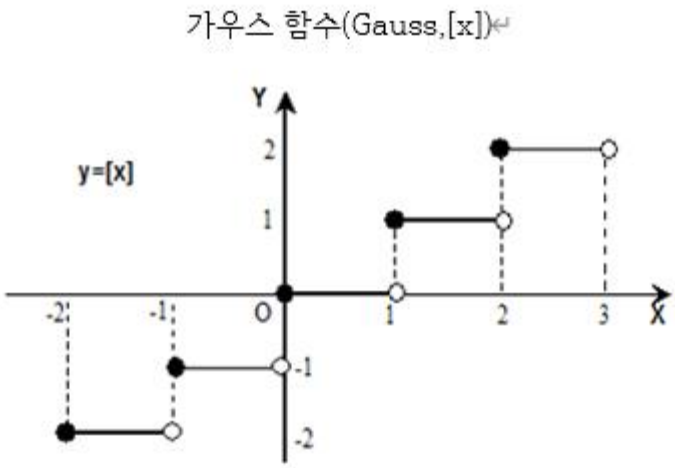
=> 표준화된 모멘텀 값에 low pass filter를 적용한 그래프이다. 단순히 필터만 적용한 그래프이므로 x축과 y축 그리고 그래프 형태의 변화는 없으며, 노이즈가 제거되어 그래프가 훨씬 매끄러워졌음을 알 수 있다.

[4] 모멘텀 신호 양자화

① 이론 설명

양자화란 연속적으로 보이는 양을 자연수로 셀 수 있는 양으로 재해석하는 것을 말한다. 아날로그 데이터를 디지털 데이터로 바꾸어 근사하는 과정을 뜻한다.

이번 프로젝트에선 양자화 과정을 가우스 함수(Gauss, [x])로 나타내었는데, 가우스 함수는 함수의 값을 정수배로 표현하여 불연속적인 값을 나타내는 함수이다. 양자화된 모멘텀 값 수식은 아래와 같다.



$$m_q[i] = Gauss((m_f[i] + Q_{bias})/Q_{level}) \times Q_{level}$$

m_q : 양자화된 모멘텀 값

m_f : 필터링된 모멘텀 값

Q_{level} : 양자화 시 나누는 단계

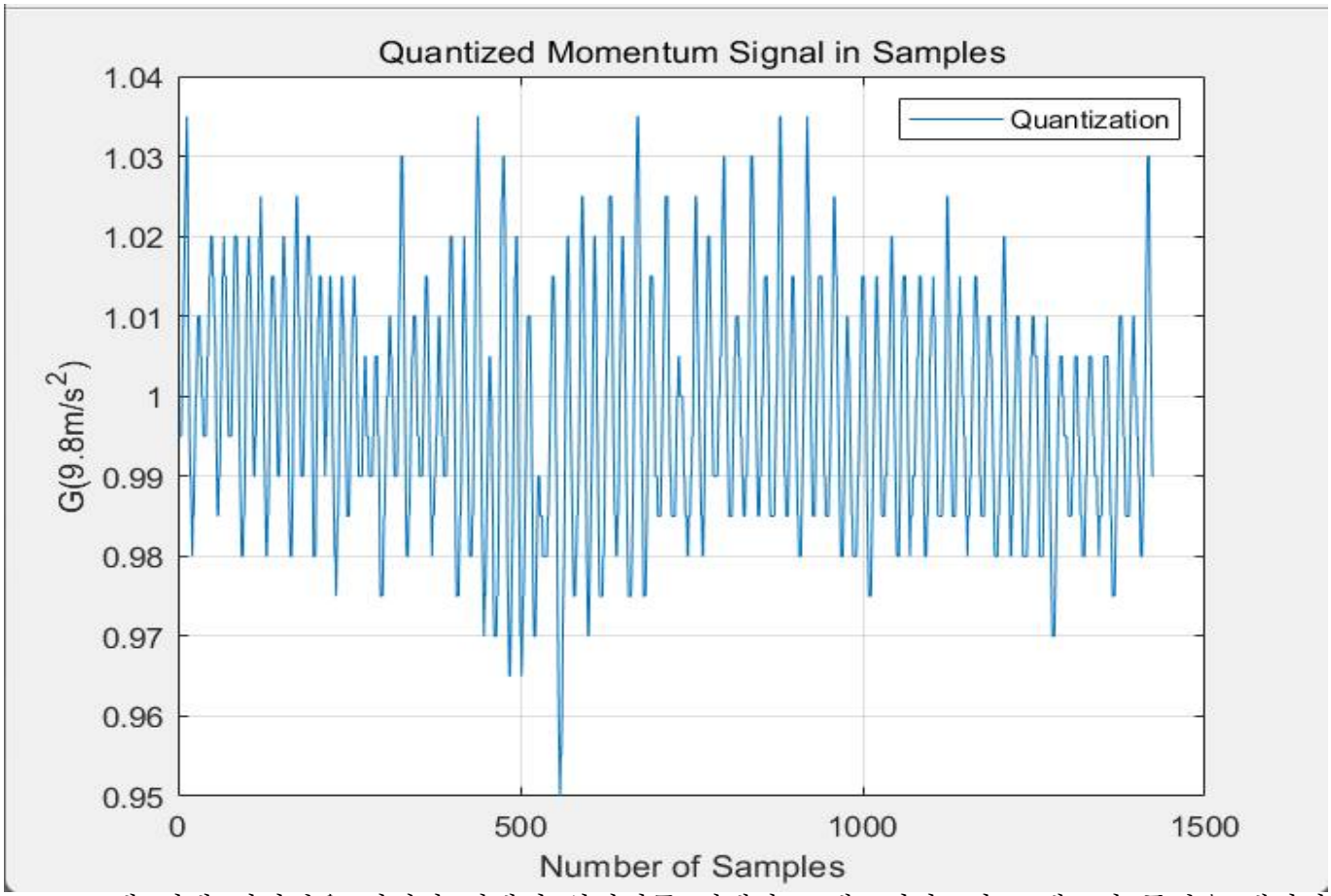
Q_{bias} : 양자화 시 [0.9 0]의 값들을 0으로 만들어주기 위해 더해주는 값

가우스 함수를 나타내기 위해 Matlab에서 floor 함수를 사용하였으며, Q_{level}은 0.005, Q_{bias}는 1로 설정하였다.

② 코드

```
%% 모멘텀 신호 양자과정 (가우스함수)
Q_level = 0.005; % Q_level 지정
Q_bias = 1; % Q_bias 지정
m_quantization = floor((momentum_filter + Q_bias) / Q_level) * Q_level;
% 양자화 공식 적용

figure()
hold on; grid on; box on;
plot(m_quantization); % 양자화된 데이터 그래프 플롯
legend('Quantization') % 범례 지정
xlabel('Number of Samples') % x축 지정
ylabel('G(9.8m/s^2)') % y축 지정
title('Quantized Momentum Signal in Samples') % 제목 지정

```

③ 그래프



=> LPF에 의해 필터링을 거쳐간 값에서 양자화를 진행한 그래프이다. 이 그래프의 특징은 데이터의 값이 단계적으로 구분되어 있다는 점이다. Matlab에서 Q_{level} 값을 0.005로 선언하였기 때문에 데이터 값이 1을 기준으로 0.005 단위씩 구분되어 있으며, 이는 양자화 과정을 통해 아날로그 데이터에서 디지털 데이터로 표현 방식이 성공적으로 바뀌었음을 알 수 있다.

[5] 피크/밸리(Peak/Valley) 검출

① 이론 설명

양자화 된 신호로부터 걸음 수를 검출하기 위해선 피크와 밸리 포인트를 이용한다.

피크/밸리 포인트를 검출하기 위해서 S, P라는 값을 이용한다.

$$S[i] = \begin{cases} 1 & m_q[i] > m_q[i-1] \\ 0 & m_q[i] < m_q[i-1] \\ S[i-1] & otherwise \end{cases}$$

$$P[i] = \begin{cases} peak & S[i] > S[i-1] \\ valley & S[i] < S[i-1] \\ none & otherwise \end{cases}$$

S는 상태를 나타내는 값으로 양자화 된 데이터에서 현재값과 이전값을 비교하여 상태를 결정한다.
P는 해당 지점의 값이 피크/밸리/해당 없음을 나타내는 값으로, S의 현재값과 이전값을 비교하여 결정한다.

② 코드

```
%% Peak / Valley 검출

S = zeros(length(m_quantization),1); % S를 0으로 초기화
P = zeros(length(m_quantization),1); % P를 0으로 초기화

for i=1:length(m_quantization)-1 % S 구하기
    if m_quantization(i) < m_quantization(i+1) % 이전값 < 현재값
        S(i) = 1;
    elseif m_quantization(i) > m_quantization(i+1) % 이전값 > 현재값
        S(i) = 0;
    else % otherwise
        S(i+1) = S(i); % 이전값 대입
    end
end

for i = 2:length(S) % P 구하기
    if S(i-1) ~= S(i)
        if S(i-1) == 1 % S[i] > S[i-1]
            P(i) = 1; % Peak
        elseif S(i-1) == 0 % S[i] < S[i-1]
            P(i) = 0; % Valley
        end

        if i > 4 % 예외처리
        if (P(i) == 0) && (P(i-1) == 1)
            P(i) = NaN;
            P(i-1) = NaN;
        elseif (P(i) == 0) && (P(i-2) == 1)
            P(i) = NaN;
            P(i-2) = NaN;
        elseif (P(i) == 0) && (P(i-3) == 1)
            P(i) = NaN;
            P(i-3) = NaN;
        end
        end
    else
        P(i) = NaN; % 무효값 처리
    end
```

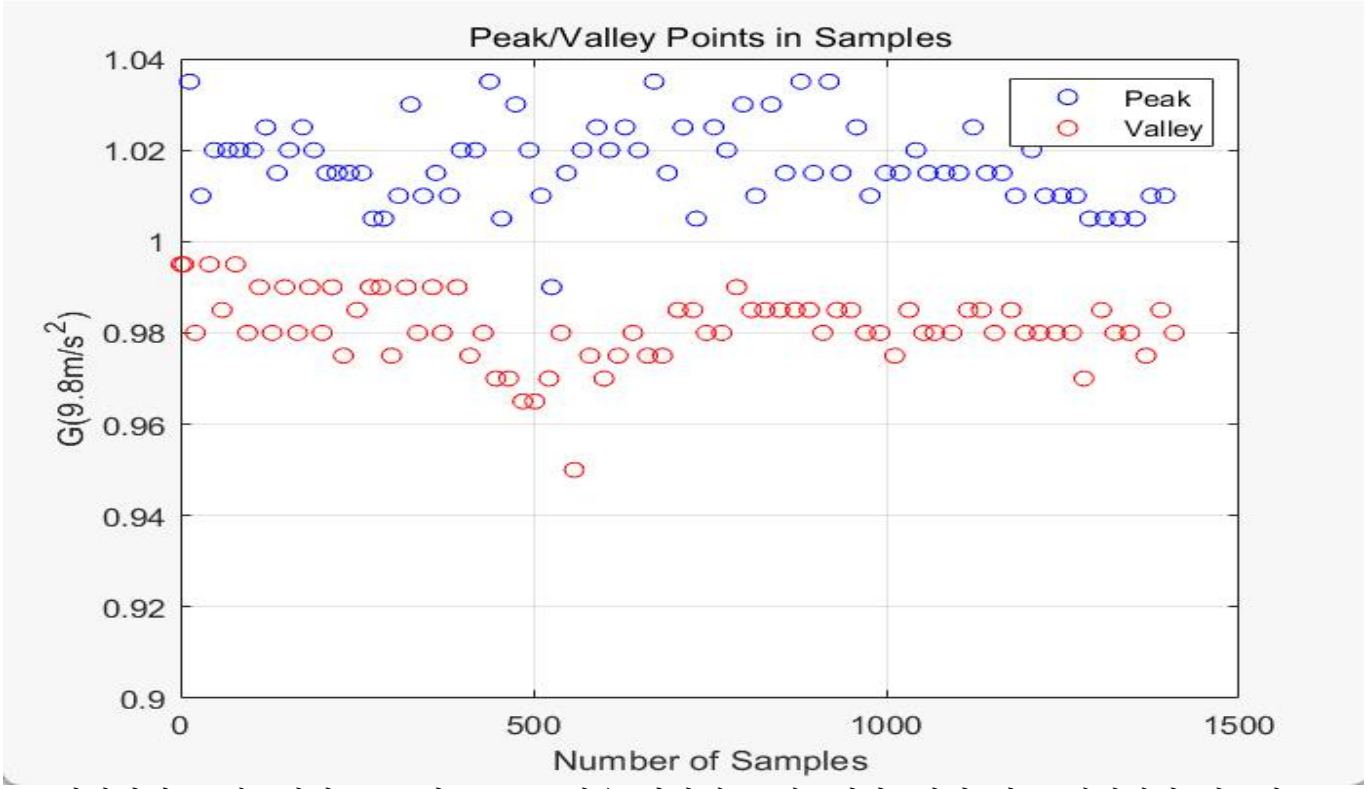
```
132     end
133
134     Peak = P; Valley = P; % peak, valley 길이 = 양자화된 데이터 길이
135
136     figure()
137     hold on; grid on; box on;
138     plot(find(P==1), m_quantization(find(P==1)),'ob') % Peak
139     plot(find(P==0), m_quantization(find(P==0)),'or') % Valley
140     ylim([0.9 1.04]) % y축 범위 조정
141     legend('Peak','Valley') % 범례 지정
142     xlabel('Number of Samples') % x축 이름 조정
143     ylabel('G(9.8m/s^2)') % y축 이름 조정
144     title('Peak/Valley Points in Samples') % 제목 조정
145
```

=> 99번째 ~ 107번째 코드는 상태 S를 구하는 과정으로서, 양자화된 데이터의 현재값과 이전값을 if문을 통해 비교하여 상태 S의 값을 정하였다.

=> 109번째 ~ 132번째 코드는 P를 구하는 과정으로서, 상태 S의 현재값과 이전값을 if문을 통해 비교하여 결정하였다. 여기서 봐야할 점은 2가지가 있다. 먼저 111번째 줄과 113번째 줄 조건에서 반복문 변수 I의 초기값을 2로 정해주었기 때문에 S(i) > S(i-1)의 조건을 "S(i-1) == 1"로 잡아주었고 S(i) < S(i-1)의 조건을 "S(i-1) == 0"으로 잡아주었다.

다음으로 예외 처리인데, 앞서 양자화된 데이터 그래프의 중간 지점에서 몇 개의 짧은 굴곡이 발생한 부분이 있었다. 이로 인해 그래프의 중간 지점에서도 peak와 valley로 표기되어 오류가 발생했었다. 따라서 117번째~128번째 코드의 예외 처리를 통해 이 문제점을 제거하였다.

③ 그래프



=> 양자화된 그래프에서 peak와 valley 값을 나타낸 그래프이다. 앞서 이론 설명에서 언급한 S, P 구하는 과정에서 피크 및 밸리를 구분하였으나, 본래 나와야 할 피크와 밸리의 개수보다 약간 적은 것을 알 수 있다. 이는 앞서 지금까지 데이터의 전처리와 양자화를 거치면서 발생한 작은 오류로 인

해 peak와 valley를 인식하지 못하는 상황이 발생한 것 같다. 본래 출력되어야 할 전처리 및 양자화 그래프와 본인의 그래프를 비교하면 몇 개 부분의 값이 약간 다른 것을 알 수 있다. 결국 이러한 부분들이 peak와 valley를 인지하지 못하여 제대로 출력하지 못한 부분이라고 생각한다.

[6] 걸음 수 검출

① 이론 설명

현재의 peak 값, 이전의 peak 값, 현재 peak 값과 이전 valley 값의 차이, peak 사이의 거리를 통해 '걸음 판단 흐름도'를 작성한다. 흐름도를 모두 통과하면 걷기 행동이라 판단하며, 걷기 행동이라 판단된 점들을 모두 더하여 걸음 수를 검출한다.

• 사용할 THRESHOLD

- THRESHOLD_ACTIVE

: 걸음이라고 판단되는 피크값의 최솟값으로 임의로 정해준다.

- THRESHOLD_SWING

: peak와 valley의 차이값이다. 해당 값이 너무 클 경우는 걸음으로 측정이 되지 않고, 너무 작을 경우는 노이즈까지 걸음으로 측정하게 되어 적절히 분리할 수 있는 값을 찾아야 한다.

- THRESHOLD_INTERVAL_LONG_TIME

: peak와 valley 사이 시간 간격의 최댓값

- THRESHOLD_INTERVAL_SHORT_TIME

: peak와 valley 사이 시간 간격의 최솟값

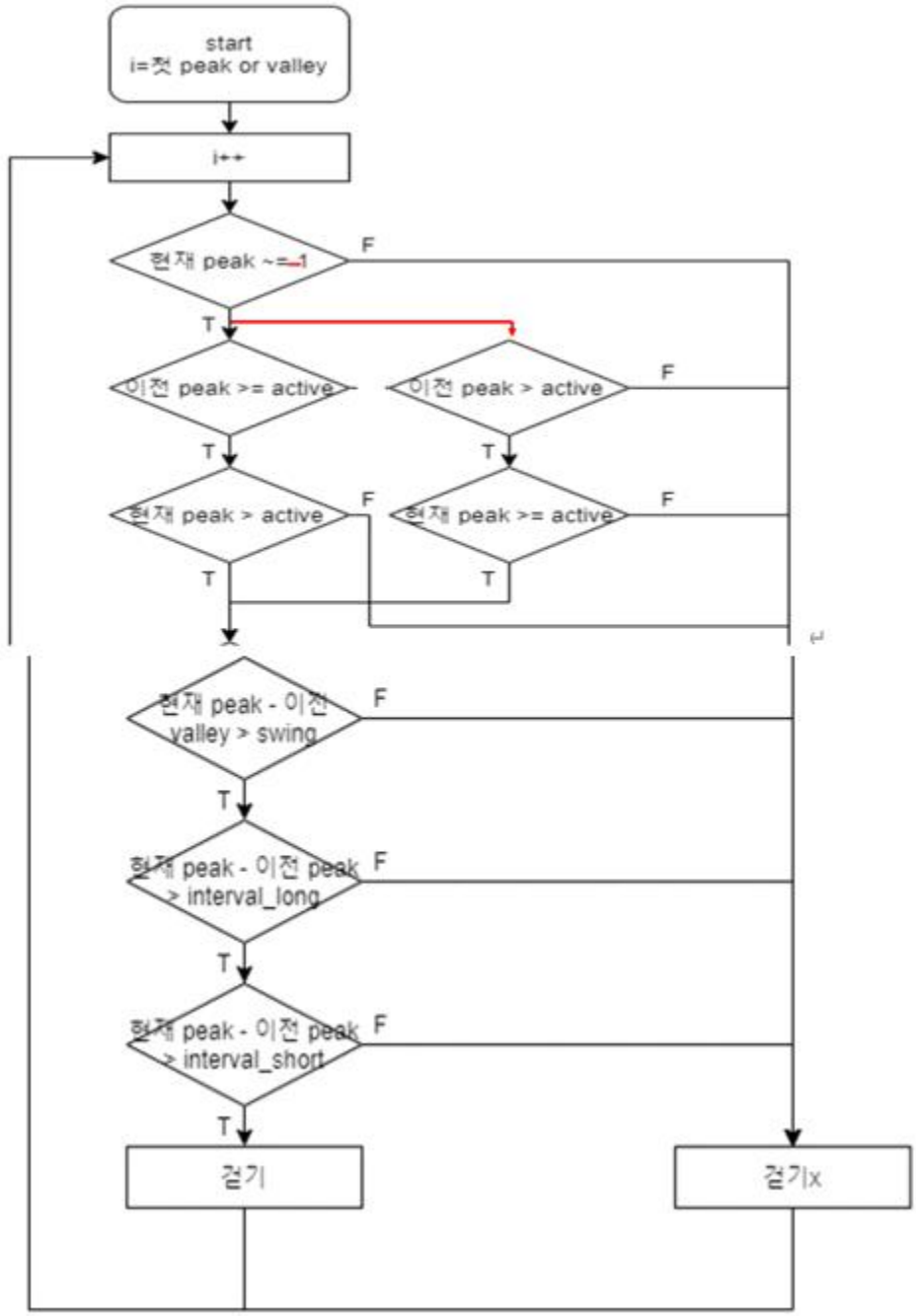
-THRESHOLD_INTERVAL_LONG_SAMPLE

: LONG_TIME과 SAMPLE RATE를 곱하여 반올림한 정수값

- THRESHOLD_INTERVAL_SHORT_SAMPLE

: SHORT_TIME과 SAMPLE_RATE를 곱하여 반올림한 정수값

• 걸음 수 추출 알고리즘


start
i=첫 peak or valley
i++
현재 peak ~=-1
F
T
이전 peak >= active
이전 peak > active
F
T
T
현재 peak > active
현재 peak >= active
F
F
T
T
현재 peak - 이전
valley > swing
F
T
현재 peak - 이전 peak
> interval_long
F
T
현재 peak - 이전 peak
> interval_short
F
T
걸기
걸기x

② 코드

```
%% 걸음수 검출

Peak_value = m_quantization(find(P==1)); % peak 값 산출
Valley_value = m_quantization(find(P==0)); % valley 값 산출

for i = 1:length(Peak_value) % peak와 valley 차이값 계산
    distance(i) = Peak_value(i) - Valley_value(i);
end

THRESHOLD_SWING = mean(distance)*0.5;  % THRESHOLD_SWING

P_index = find(P==1); % peak인 점의 index 대입
V_index = find(P==0); % valley인 점의 index 대입

THRESHOLD_ACTIVE = 1; % THRESHOLD_ACTIVE, 임의 조정

THRESHOLD_INTERVAL_TIME = zeros(length(P_index), 1);
for i = 1:length(P_index) % peak와 valley의 시간 간격
    THRESHOLD_INTERVAL_TIME(i) = P_index(i) - V_index(i);
end

THRESHOLD_INTERVAL_LONG_TIME = max(THRESHOLD_INTERVAL_TIME);
% peak와 valley 사이 시간 간격 최댓값
THRESHOLD_INTERVAL_SHORT_TIME = min(THRESHOLD_INTERVAL_TIME);
% peak와 valley 사이 시간 간격 최솟값
THRESHOLD_INTERVAL_LONG_SAMPLE = round(THRESHOLD_INTERVAL_LONG_TIME * SAMPLE_RATE);
% THRESHOLD_INTERVAL_LONG_SAMPLE
THRESHOLD_INTERVAL_SHORT_SAMPLE = round(THRESHOLD_INTERVAL_SHORT_TIME * SAMPLE_RATE);
% THRESHOLD_INTERVAL_SHORT_SAMPLE

i = 1; k =1;
for j = 1 : length(P_index) % j = 73까지 반복문 진행
    if i < length(P_index)
    i = i + 1; % i++
    if Peak_value(i) ~= -1 % 현재 peak ~= -1
        if Peak_value(i-1) >= THRESHOLD_ACTIVE % 이전 peak >= active
            if Peak_value(i) >= THRESHOLD_ACTIVE % 현재 peak >= active
                if Peak_value(i) - Valley_value(i-1) >= THRESHOLD_SWING
                    % 현재 peak - 이전 valley >= swing
                    if P_index(i) - P_index(i-1) <= THRESHOLD_INTERVAL_LONG_TIME
                        % 현재 peak - 이전 peak <= interval_long
                        if P_index(i) - P_index(i-1) >= THRESHOLD_INTERVAL_SHORT_TIME
                            % 현재 peak - 이전 peak <= interval_short
                            if round((P_index(i) - P_index(i-1)) * SAMPLE_RATE) <= THRESHOLD_INTERVAL_LONG_SAMPLE
                                % 현재 peak - 이전 peak <= interval_long
                                if round((P_index(i) - P_index(i-1)) * SAMPLE_RATE) >= THRESHOLD_INTERVAL_SHORT_SAMPLE
                                    % 현재 peak - 이전 peak <= interval_short
                                    Step(i) = 1;
                                else
                                    Step(i) = 0;
                                    no_Peak(k) = i;
                                    k = k + 1;
                                end
                            else
                                Step(i) = 0;
                                no_Peak(k) = i;
                                k = k + 1;
                            end
                        else
                            Step(i) = 0;
                            no_Peak(k) = i;
                            k = k + 1;
                        end
                    else
                        Step(i) = 0;
                        no_Peak(k) = i;
                        k = k + 1;
                    end
                else
                    Step(i) = 0;
                    no_Peak(k) = i;
                    k = k + 1;
                end
            else
                Step(i) = 0;
                no_Peak(k) = i;
                k = k + 1;
            end
```

=> Peak_value와 Valley_value는 peak과 valley인 부분의 양자화 값만을 산출하기 위해 나타낸 배열이다. 이는 차이값을 나타내는 distance 계산과 나중에 있을 step 검출 알고리즘에 사용된다.

=> 152번째 ~ 154번째 코드 : 각 peak와 valley의 차이를 계산하는 반복문이다.

=> THRESHOLD_SWING은 distance의 평균값에 임의로 0.5를 곱한 값으로 나타내었다.

=> P_index와 V_index는 peak와 valley인 점의 index만을 저장하는 배열이고, 이는 peak와

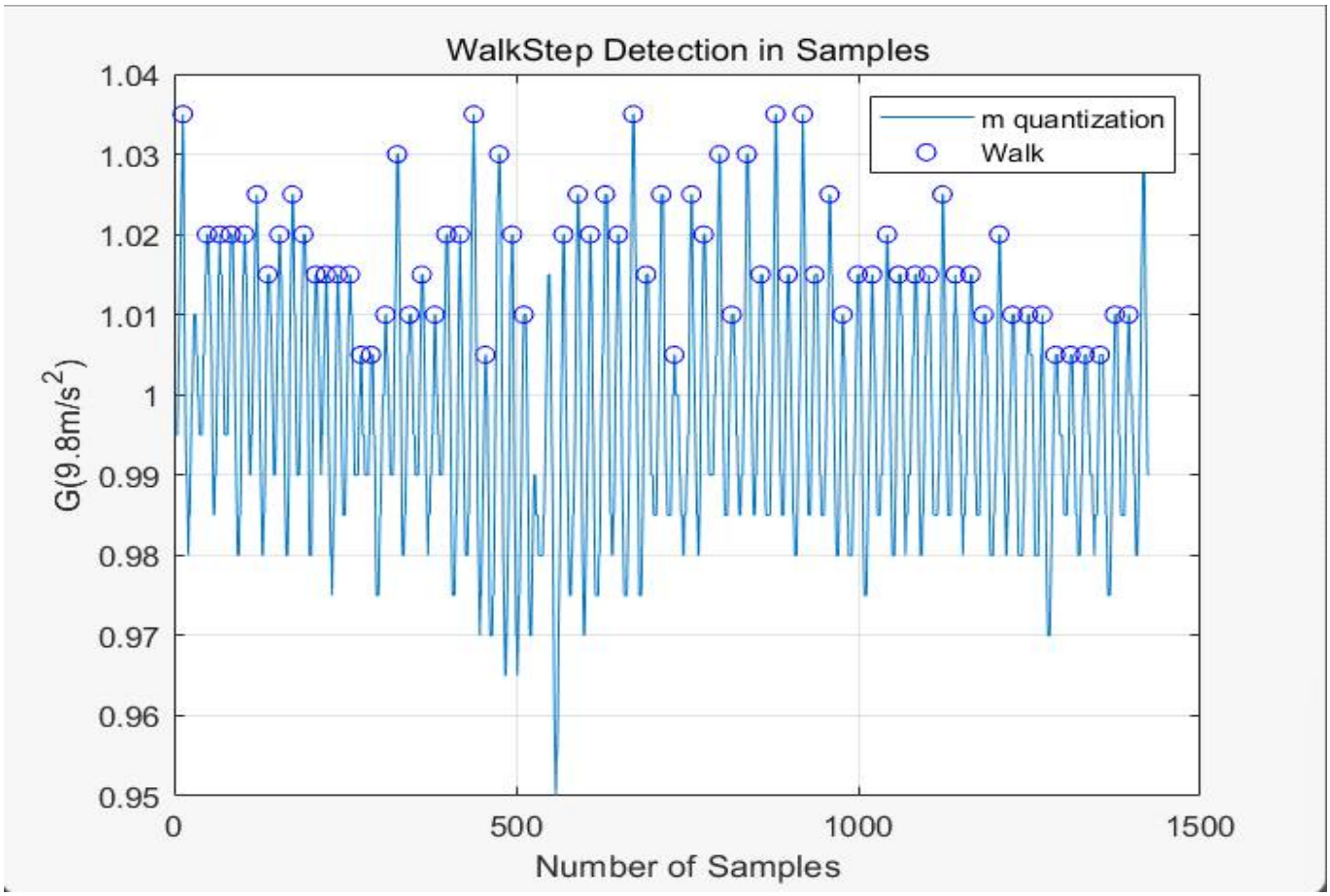
valley의 시간 간격을 계산할 때와 step 검출 알고리즘에서 사용된다.
=> THRESHOLD_ACTIVE는 임의로 값을 1로 맞추었다.
=> 164번째 ~ 166번째 코드 : 각 peak와 valley의 시간 간격을 계산하는 반복문이다.
=> THRESHOLD_INTERVAL_LONG_TIME, THRESHOLD_INTERVAL_SHORT_TIME
: 앞서 반복문으로 구한 각 peak와 valley의 시간 간격값에 max함수 및 min함수를 적용하여 최댓값과 최솟값을 구하였다. 즉 LONG_TIME은 시간 간격의 최댓값으로 34이고 SHORT_TIME은 시간 간격의 최솟값으로 12이다.
=> THRESHOLD_INTERVAL_LONG_SAMPLE, THRESHOLD_INTERVAL_SHORT_SAMPLE
: 앞서 구한 LONG_TIME과 SHORT_TIME에 SAMPLE_RATE를 곱하여 반올림한 정수값이다. 이번 프로젝트에서 SAMPLE_RATE는 32Hz를 사용하였으며, 반올림한 정수값을 구하기 위해 round 함수를 사용하였다. 결국 이 둘의 값은 각각 1088, 384로 계산되었다.

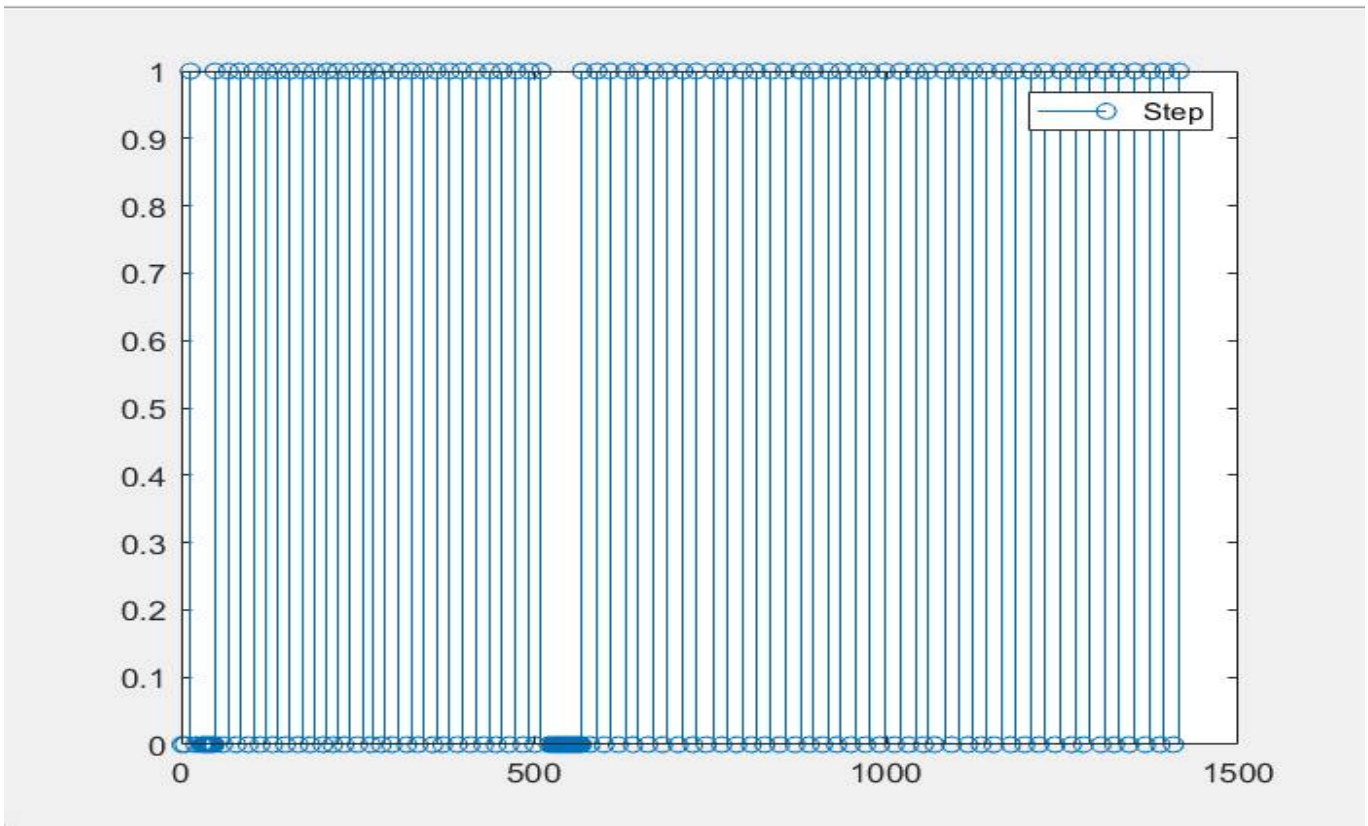
```
            else
                Step(i) = 0;
                no_Peak(k) = i;
                k = k + 1;
            end
        else
            Step(i) = 0;
            no_Peak(k) = i;
            k = k + 1;
        end
    else
        Step(i) = 0;
        no_Peak(k) = i;
        k = k + 1;
    end
  end
end

cnt=0;
for i = 1:length(P)
    if P(i) == 1
        cnt = cnt + 1;
    end
    if cnt == 2
        P(i) = 0;
    elseif cnt == 30
        P(i) = 0;
    elseif cnt == 31
        P(i) = 0;
    end
end

% 인식률 산출
N_r = 71;
N_a = sum(Step);
R = (N_r - (abs(N_r - N_a)))/N_r * 100;

disp(R); % 97.1831
disp(N_a); % 69
disp(THRESHOLD_ACTIVE); % 1
disp(THRESHOLD_SWING); % 0.0180
disp(THRESHOLD_INTERVAL_SHORT_TIME); % 12
disp(THRESHOLD_INTERVAL_LONG_TIME); % 34
disp(THRESHOLD_INTERVAL_SHORT_SAMPLE); % 384
disp(THRESHOLD_INTERVAL_LONG_SAMPLE); % 1088

figure()
hold on; grid on; box on;
plot(m_quantization)
plot(find(P==1), m_quantization(find(P==1)),'ob')
xlabel('Number of Samples')
ylabel('G(9.8m/s^2)')
title('WalkStep Detection in Samples')
legend('m quantization','Walk');

figure()
x = floor(P);
stem(x);
legend('Step');
```

=> 177번째 ~ 236번째 코드 : 걸음 수를 추출하는 알고리즘이며, 위에 있는 이론 설명에서 언급한 block diagram 순서대로 진행하였다. no_Peak 배열은 걸음이 아닌 peak일 때 index 값을 골라내

기 위하여 만든 배열이고 2, 30, 31의 index가 추출되어 세 개의 값이 이 배열에 저장된다.
=> 238번째 ~ 250번째 코드 : 피크, 밸리, 해당없음이 저장된 P배열에서 앞서 구한 2번째, 30번재, 31번째 peak의 값을 0으로 만들어주는 반복문이다. 이 반복문으로 step 검열 알고리즘에서 step이 아닌 것으로 결정된 peak를 peak가 아닌 다른 값으로 변경한다.
=> 253번째 ~ 255번째 코드 : 인식률을 계산하는 코드이며, N_a는 지금까지 구한 step에서 matlab의 sum함수를 이용하여 계산하였다. 해당 코드를 통해 계산된 인식률은 97.1831%이다.

③ 그래프





=> 최종판단 걸음을 나타낸 그래프이며, 걸음 판단 알고리즘을 거쳐 조건에 맞지 않는 peak 값은 제외하였다. 본래 출력되어야 할 그래프와 비교하면 극소수의 peak 부분이 나타나지 않을 것을 볼 수 있는데, 신호에 대해서 알고리즘이 추정한 횟수 N_a가 69이므로 총 3개의 peak가 누락되었다고

할 수 있다. 이는 THRESHOLD_TIME 또는 THRESHOLD_SAMPLE 부분에서 잘못 걸러진 것으로 생각하며, 출력되지 않은 step보다 낮은 값의 step이 존재하는 것이 그 이유이다. ACTIVE 또는 SWING으로 값을 조절하여 출력할 수 있는 것이 아니라고 생각한다. 초반 전처리 및 양자화의 그래프에서도 본래 나와야 할 그래프와 약간의 차이가 존재했기 때문에, 그 부분에서 발생한 문제일 가능성이 크다. 하지만 인식률이 약 97%이므로 매우 성공적인 알고리즘을 구성하였다고 할 수 있다. 특히 프로그램 경과 시간은 약 10.32초가 발생했는데, 조금 지체된 시간이지만 동일한 코드를 다른 조원의 PC로 프로그램 실행하였을 때 모두 경과 시간이 다르기 때문에 사용자의 PC 환경이 느린 경과 시간의 큰 원인이라고 생각한다.

※ 출력화면

```
    97.1831

     69

      1

    0.0180

     12

     34

    384

         1088

경과 시간은 10.319734초입니다.
fx >>
```

위에서부터 순서대로
인식률,
측정 걸음 수,
THRESHOLD_ACTIVE,
THRESHOLD_SWING,
THRESHOLD_INTERVAL_SHORT_TIME,
THRESHOLD_INTERVAL_LONG_TIME,
THRESHOLD_INTERVAL_SHORT_SAMPLE,

THRESHOLD_INTERVAL_LONG_SAMPLE,
프로그램 경과 시간이다.