Tool Wear Diagnosis Device Development Manual

B635385 정석준**(wjdtjrwns97@naver.com)**

B635459 한태연**(hgf2970@naver.com)**

B817123 정승원**(sw1598@naver.com)**

Index

Sensing - LabVIEW

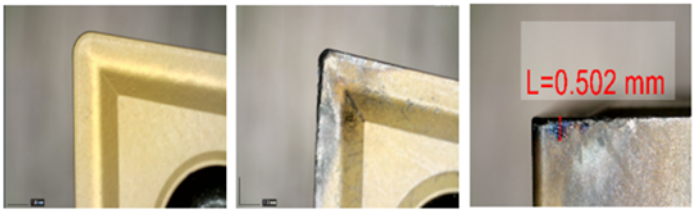
Experimental Conditions

현장에서 하나의 가공품을 제작하기 위해 여러 가공조건이 연속적으로 실행되는 것을 반영하기 위해 가공조건에 변화를 주어 3단계로 이루어진 일련의 공정을 설정하여 실험을 진행하였습니다.

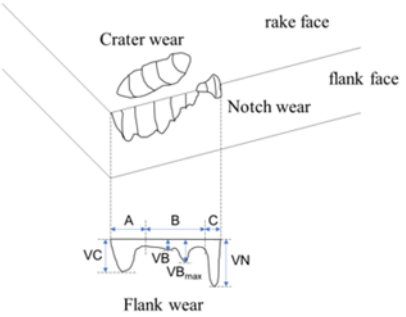
선반은 화천기계의 HL-380, 피삭제는 S45C 원통형 연강 시편, 인서트는 초경 공구인 CNMG120408B25-NC3030을 사용하였습니다.

공구는 정상과 마모로 구분하여 사용하였으며 여유면의 평균마모폭이 0.3mm이상이거나 최대마모폭이 0.5mm인 경우 마모공구로 판단하여 사용하였습니다.

$VB = 0.3mm$ 이상 or $VB_{max} = 0.5mm$ 이상[1)]

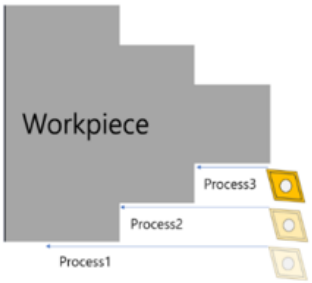


정상(Rake face) 마모(Rake face) 마모(Flank face)



회전속도, 이송률, 절삭깊이 등의 가공조건은 표와 같으며 우측 그림과 같이 3번의 행정이 연속적으로 이루어집니다.

	Spindle speed(rpm)	Feed rate(mm/rev)	Depth of cut(mm)	Length (mm)
1st	900	0.33	2	25
2nd	900	0.24	2	20
3rd	900	0.24	3	15



마모 공구를 사용하기 때문에 안전에 유의하여 가혹한 가공조건은 피해서 진행하였습니다.

Sensor Selection

저희는 진동과 소음 데이터를 분석하기로 결정하였으며 이를 위해 적절한 센서를 선정하였습니다. 센서는 개발할 기기에서 사용할 저비용의 센서와 해당 센서들의 신뢰도 확인을 위해 비교할 고성능 센서를 사용하였습니다.

저비용의 센서는 마이크와 피에조 센서를 대역폭, 민감도, 입력 전압, 비용 등을 고려하여 선정하였습니다. 고성능 센서는 비교를 위해 마이크, 가속도계, AE 센서를 사용하였으며 데이터 분석을 위한 DAQ는 NI사의 NI-9775를 사용하였습니다.

선정된 센서 정보는 다음과 같습니다.

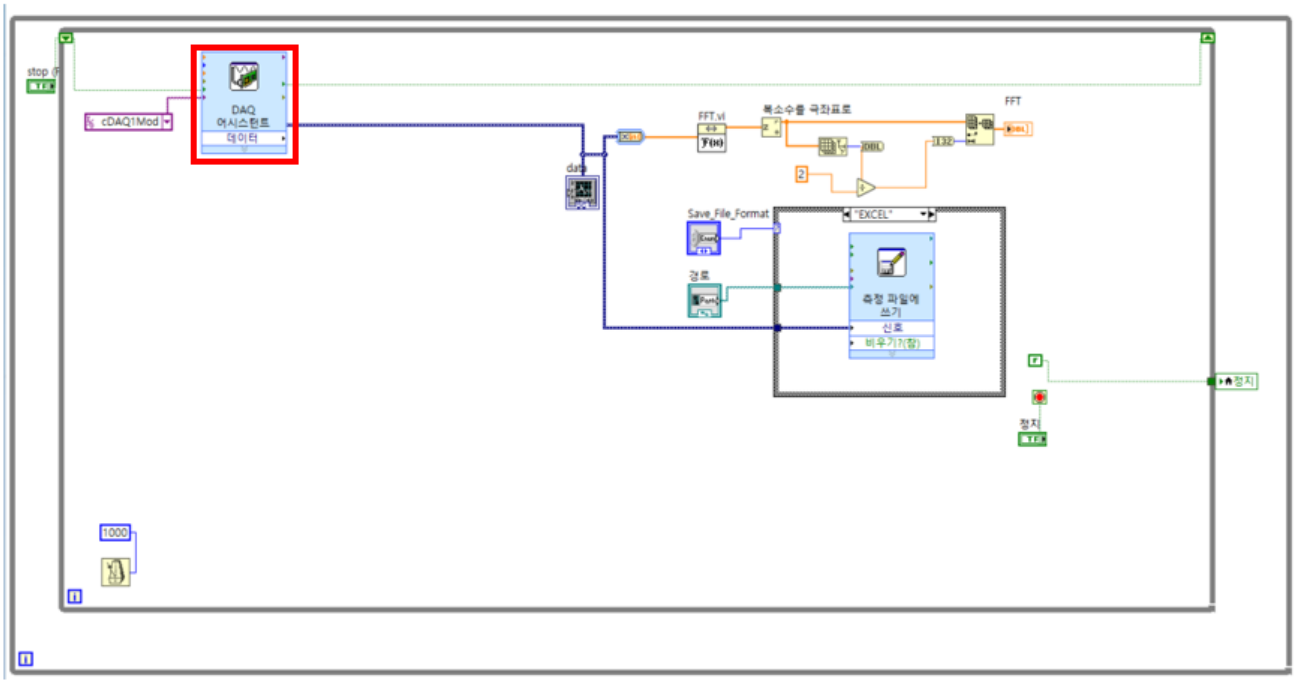
고성능	모델명	저비용	모델명
AE 센서	AE Sensor/R15a	피에조센서	DFR0052
마이크로폰	CRY333-T1	마이크로폰	GY-MAX9814
가속도계	AC214-1D		

LabVIEW

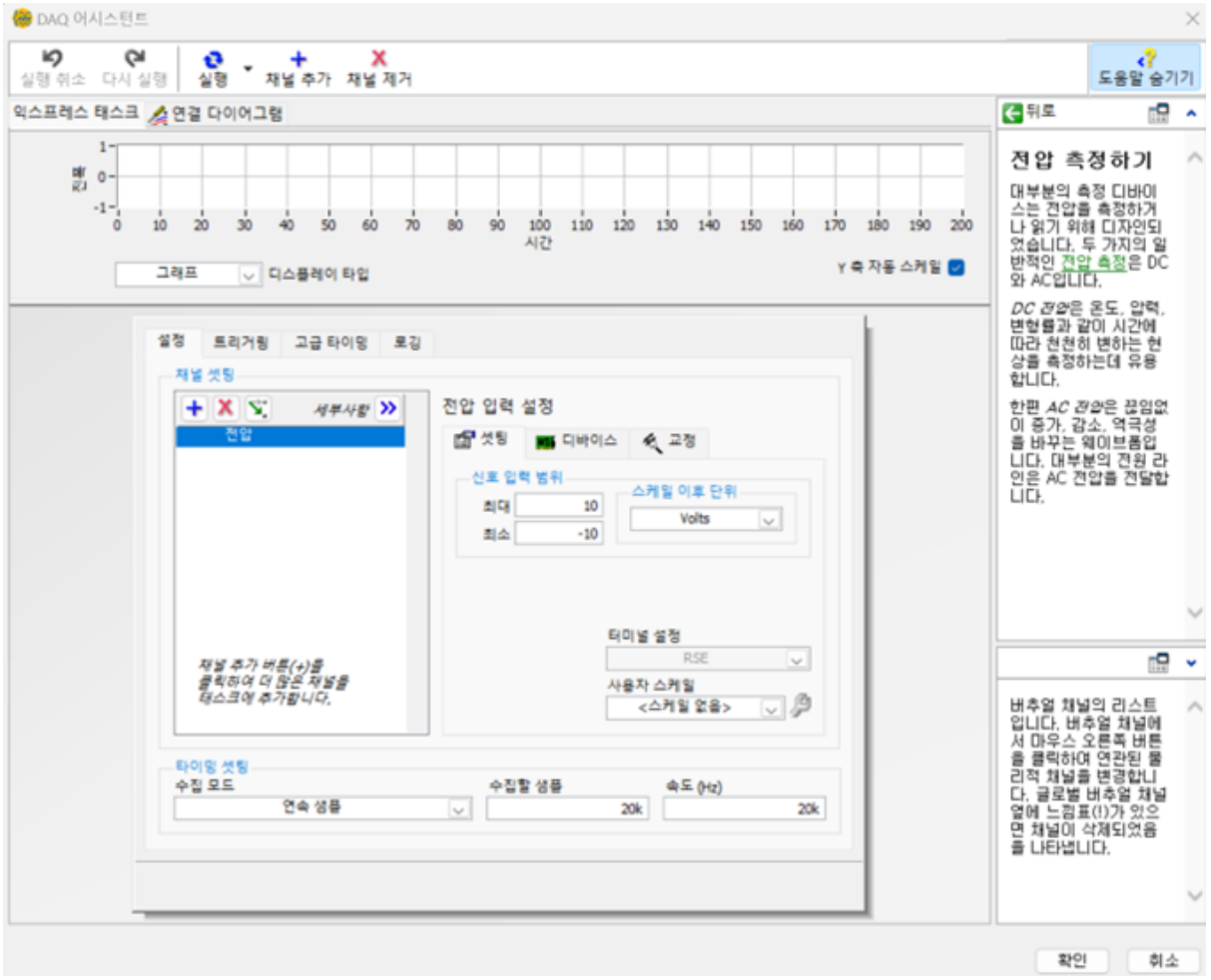
고성능의 가속도계와 마이크 센서, AE센서를 사용하는 경우에 DAQ를 사용하면, 원하는 주파수 만큼 수집할 수 있습니다. AE센서의 실험 결과 약 50kHz까지는 무리 없이 수집할 수 있었으나, 그 이상의 주파수를 측정하는 경우에 오류가 종종 발생합니다.

추가로, 가속도계의 한계가 10kHz인 경우더라도 아래 수집 속도를 수정하며 10kHz이상의 주파수를 측정할 수 있으나, 이는 데이터의 신뢰도를 저하하므로 센서의 성능에 맞게 수집 속도를 선정하는 것이 필수적입니다.

LabView는 링크를 참고하여 코드를 작성하였습니다. 실험에 사용된 코드 파일은 'plz'로 저장되어있으며 아래는 그 사진입니다.



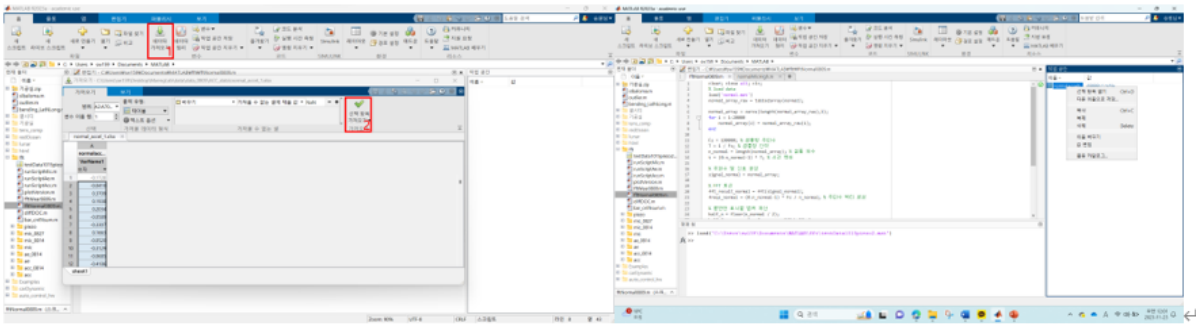
이 파일에서 DAQ의 수집속도를 변경하고 싶다면, 빨간색 박스를 더블클릭 시, 다음과 같은 화면을 볼 수 있습니다.



위의 빨간 네모에서 편집할 수 있으며, 수집 모드는 '연속 샘플'로 고정한 후, 수집할 샘플과 속도를 1:1 비율로 맞춰서 사용하면 됩니다. 예를 들어, 20k, 20k로 설정할 경우 10kHz까지 측정할 수 있으며, 40k, 40k로 설정할 경우 20kHz까지 측정할 수 있습니다.

Matlab Code

위의 LabView를 통해 데이터를 수집하게 되면, .csv 혹은 .xlsx 파일로 저장해 놓았을 것입니다. 그렇다면 Matlab에 다음과 같은 과정으로 '.mat' 형식의 파일로 저장하는 작업이 필요합니다.



우측 그림에서 다른 이름으로 저장 후, 저장하고 싶은 파일명을 입력합니다. 저장한 파일로 아래 첨부한 코드로 주파수 도메인으로 변경할 수 있습니다.

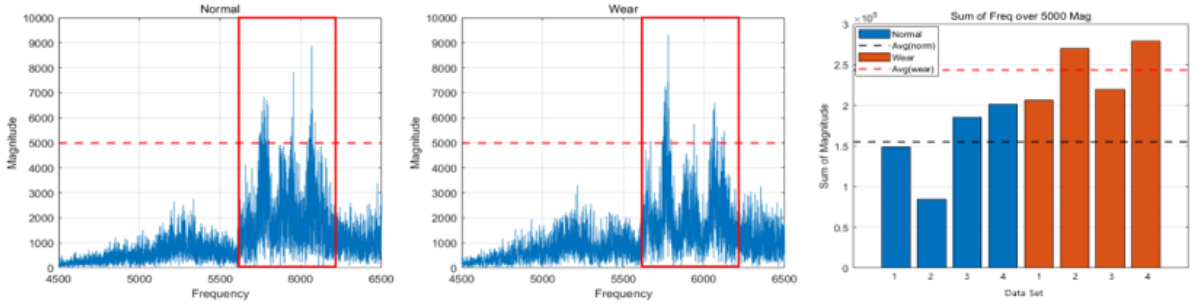
```
clear; close all; clc;

% load data

load('normal.mat')

normal_array_raw = table2array(normal);

normal_array = zeros(length(normal_array_raw),1);

for i = 1:20000

        normal_array(i) = normal_array_raw(i);

end

Fs = 100000; % 샘플링 주파수

T = 1 / Fs; % 샘플링 간격

n_normal = length(normal_array); % 샘플 개수

t = (0:n_normal-1) * T; % 시간 벡터

% 주파수 및 신호 생성

signal_normal = normal_array;

% FFT 계산

fft_result_normal = fft(signal_normal);

freqs_normal = (0:n_normal-1) * Fs / n_normal; % 주파수 벡터 생성

% 절반만 표시할 범위 계산

half_n = floor(n_normal / 2);

half_freqs_normal = freqs_normal(1:half_n);

half_fft_result_normal = fft_result_normal(1:half_n);

% 결과 그래프 그리기

figure(1);

plot(half_freqs_normal, abs(half_fft_result_normal));

grid on;

title('Frequency Domain (FFT)');

xlabel('Frequency (kHz)');

xlim([0 5000])

ylim([0 3000])
ylabel('Magnitude');
```

이제, 원하는 주파수 영역대에서 일정 Magnitude가 넘는 코드를 첨부합니다.

```
% Setup

minFreq = 4000; % min freq which i want to see

maxFreq = 6000; % max freq which i want to see

freqLength = (maxFreq - minFreq)*2; % LabView gives data each 0.5 Freq

freq6000Hz = zeros(freqLength, 1);

result6000Hz1 = zeros(freqLength, 1);

result6000Hz2 = zeros(freqLength, 1);

result6000Hz3 = zeros(freqLength, 1);

result6000Hz4 = zeros(freqLength, 1);

% data extract (normal)

t = 1;

for i = 1:length(half_fft_result_normal1)

        if half_freqs_normal(i) >= minFreq && half_freqs_normal(i) <= maxFreq % 5500~6500

        freq6000Hz(t) = half_freqs_normal(i);

        result6000Hz1(t) = half_fft_result_normal1(i);

        result6000Hz2(t) = half_fft_result_normal2(i);

        result6000Hz3(t) = half_fft_result_normal3(i);

        result6000Hz4(t) = half_fft_result_normal4(i);

        t = t + 1;

        end

end

% sum over 5000 Mag

mag = 5;

sum1 = 0; cnt1 = 0;

sum2 = 0; cnt2 = 0;

sum3 = 0; cnt3 = 0;

sum4 = 0; cnt4 = 0;

for i = 1:freqLength

        if abs(result6000Hz1(i)) > mag

        sum1 = sum1 + abs(result6000Hz1(i));

        cnt1 = cnt1 + 1;
```

```
        elseif abs(result6000Hz2(i)) > mag
        sum2 = sum2 + abs(result6000Hz2(i));
        cnt2 = cnt2 + 1;
        elseif abs(result6000Hz3(i)) > mag
        sum3 = sum3 + abs(result6000Hz3(i));
        cnt3 = cnt3 + 1;
        elseif abs(result6000Hz4(i)) > mag
        sum4 = sum4 + abs(result6000Hz4(i));
        cnt4 = cnt4 + 1;
        end
end
```

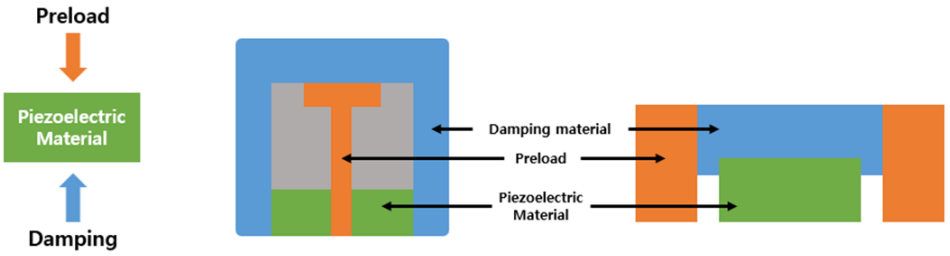
위 코드를 실행하게 되면, 다음과 같은 결과를 얻을 수 있습니다.



Sensing - Raspberry pi

Sensor Selection

저비용의 가속도계는 공구의 정상과 마모 차이를 확인한 약 6000Hz대에서 민감한 반응을 하지 못했습니다.이에 압전 소자를 이용한 센서를 고안했고, FBD는 다음과 같습니다. Damping Material로는 구하기 쉬운 부직포를 사용했고, 수직하중은 자석을 이용했습니다(Model: DFR0052).



마이크 센서의 경우 10000Hz~15000Hz에서 차이를 충분히 구분할 수 있었습니다(Model: GY-MAX9814).

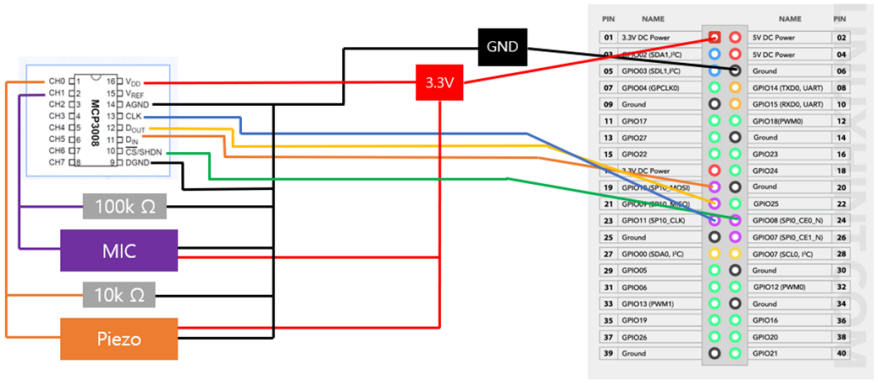
Arduino Board

Arduino는 다양한 방법을 시도해 보았으나, 1초에 약 1000개의 데이터밖에 수집하지 못했습니다. 따라서 500Hz까지 밖에 측정할 수 없었습니다. 이는 RAM에 저장한 후 파일로 저장하는 과정에서 시간을 소요하는 이유로 생각됩니다.

그렇기에 저희 팀은 Clock speed와 RAM 저장이 4GB까지 가능한 Raspberry pi 4B를 사용하여 데이터를 수집하고 저장했습니다.

Raspberry pi

Raspberry pi는 input 데이터로 digital 데이터를 받습니다. 센서는 Analog 형식으로 데이터를 출력하므로 ADC가 필요했고, 본 연구에서는 MCP3008을 활용했습니다.



마이크 센서와 피에조 센서의 저항으로는 각각 100k옴, 10k옴 일 때 데이터가 가장 잘 나왔습니다.

Raspberry pi의 높을 Clock speed를 활용하고 가능한 높은 주파수 대를 측정하기 위해, 수집 코드에는 Multi-threading을 사용했습니다. 해당 코드를 라즈베리 파이에 입력하는 방법은 아래 코드 이후에 기재하였습니다.

```
import time

import csv

from gpiozero import MCP3008

import ctypes

from threading import Thread

channel_piezo = 0

channel_mic = 1

libc = ctypes.CDLL('libc.so.6')

start_time = time.time()

end_time = start_time + 2

piezo = MCP3008(channel=channel_piezo)

mic = MCP3008(channel=channel_mic)

piezo_list = []

mic_list = []

def usleep(microseconds):

        libc.usleep(microseconds)

def piezoFun:

        while time.time() < end_time:

        value_piezo = round(piezo.value, 6)

        piezo_list.append(value_piezo)

def micFun:

        while time.time() < end_time:

        value_mic = round(mic.value, 6)

        mic_list.append(value_mic)

if __name__=='__main__':

        proc1 = Thread(target=piezoFun, args=())

        proc2 = Thread(target=micFun, args=())

        proc1.start()
```

```
proc2.start()

proc1.join()

proc2.join()

with open('test_data_normal.csv', 'w', newline='') as csvfile:

fieldnames = ['Piezo Value', 'Mic Value']

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

writer.writeheader()

for piezo, mic in zip(piezo_list, mic_list):

writer.writerow({'Piezo Value': piezo, 'Mic Value': mic})

print("Data collection and CSV writing complete.")
```

라즈베리 파이 사용법에 대한 간략한 설명입니다.

우선, 해당 링크에서 라즈베리 파이의 운영 체제를 사용하는 환경에 맞게 초기화 작업을 해야 합니다. 아래 영상들을 따라간다면 GUI를 사용할 수 있습니다. 그러나 본 연구에서는 PuTTY와 VNC viewer의 연결 불안정성 때문에 이를 사용하지 않고, cmd 창에서 ssh 연결을 활용했습니다.

링크 아래는 그 연결과 조작 방법에 대한 설명을 기재하였습니다.

Link 1

Link 2

ssh pi@192.168.2.231 (pi라는 이름과 192.168.2.231이라는 IP주소에 ssh 연결을 한다는 명령어)

이후, password를 입력하면 됩니다. Password는 입력해도 cmd창에 출력되지 않으므로 당황하지 않아도 됩니다.

여기까지 완료하면, PC와 Raspberry pi는 연결된 상태입니다. ssh 연결은 위의 연결보다 연결 안정성이 높다는 장점이 있습니다.

작성한 코드들을 라즈베리 파이에 입력하기 위한 명령어를 아래에 정리하였습니다.

ls: 현재 디렉토리에 있는 파일을 cmd창에서 확인할 수 있는 명령어입니다.

cd: 원하고자 하는 디렉토리로 이동이 가능한 명령어입니다. 즉, ‘cd /home/pi’를 입력하면 home에 있는 pi안의 파일에 이동합니다. ‘cd ..’을 입력하면 한단계 전의 디렉토리로 이동합니다.

nano: nano라는 편집기를 사용하여 python코드를 입력할 수 있습니다. 이후, shift+x입력, y+enter을 입력하면 수정된 코드가 저장된 채로 cmd창으로 복귀할 수 있습니다.

python: 위의 저장한 python 코드 파일을 실행하는 명령어입니다.

예시.

파일 이름이 ‘delRaspberrypi’인 경우

nano deliciousRaspberrypi.py >> ‘deliciousRaspberrypi’라는 이름의 파이썬 코드 파일 열람

python deliciousRaspberrypi.py >> ‘deliciousRaspberrypi’라는 이름의 파이썬 코드 파일 실행이 가능합니다.

이후, 설계한 머신러닝 알고리즘을 Raspberry pi에 입력하려면, pickle 파일을 실행해야 합니다. 이는 raspberry pi 내에서 가상환경을 설정한 후 사용했습니다.

python3 -m venv path/to/venv: 가상환경 설정 명령어

deactivate: 가상환경 종료 명령어

이외에 추가적인 명령어를 아래에 기입하였습니다.

mv: 파일 옮기기

rm: 파일 삭제하기

pip list: 설치된 패키지 확인

pip install: python 패키지 설치

sudo: 관리자 권한 실행

tab: 명령어 자동 완성

alias: 명령어 단축키 설정 (bash 파일에서 설정 필요)

shift+x: 종료 >> 이후에 저장할 것이냐는 물음이 나오면, y를 누르면 저장한 후 종료

exit: cmd 창 종료

개인적 고찰 및 추천 방안 - 정승원

본 연구에서는 Raspberry pi의 편집기로 nano를 사용했습니다. 그러나 nano는 편의상의 어려움이 종종 있고, nano가 아닌 vim을 사용할 수도 있으며, visual studio를 연결하는 방법 또한 유튜브에 존재합니다. 따라서, 연구에 코드의 양이 방대하고 수정이 자주 필요한 경우, 다른 편집기를 사용하는 것을 추천드립니다.

또한, 파일의 저장을 내장 메모리, 외장 메모리(usb), pc에 무선 전송, 서버에 저장하는 등 다양한 방법이 있습니다. 서버에 저장하는 방법을 제외하면 ssh 연결을 통해 pc와 통신을 하게 되는데, 이는 pc와 약 5m 이상이 떨어진다면 연결이 불안정한 모습 또한 관찰할 수 있었습니다. 이에 서버에 파일을 저장하고, 그것을 pc에 내려 받는 방법이 더 유리할 것이라는 생각을 가지고 있습니다.

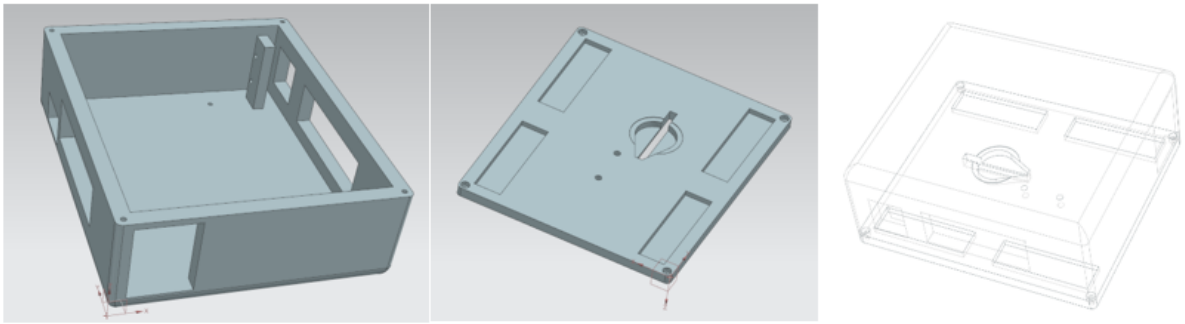
데이터를 수집시에 multi-threading을 사용하여 1초에 약 6000개의 데이터를 수집할 수 있었습니다. 코드는 while문이라고 하더라도 1번 실행되는 real-time이 존재하기에 최대한 간결한 것이 중요합니다. 저희는 CPU가 4개인 상황에서 2개의 threading을 사용했는데, raspberry pi 4B는 CPU가 4개라는 점에서 총 4개의 병렬 threading이 가능합니다. 필요하다면 이 점을 활용해보길 바랍니다.

마지막으로, pc와 연결시, GUI를 사용하는 방법은 직관적으로 이해하기 쉬운 방법입니다. 그러나 GUI는 연결 불안정으로 많은 시간을 소요하므로, 처음 조금 어렵더라도 ssh연결을 추천드립니다.

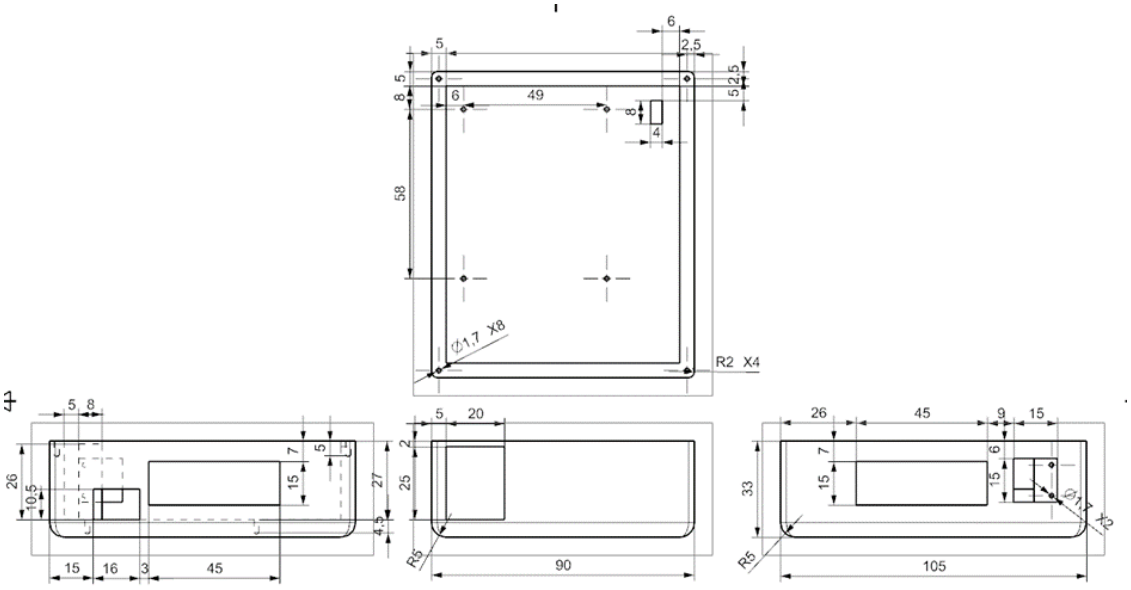
Module Design Detailed Information

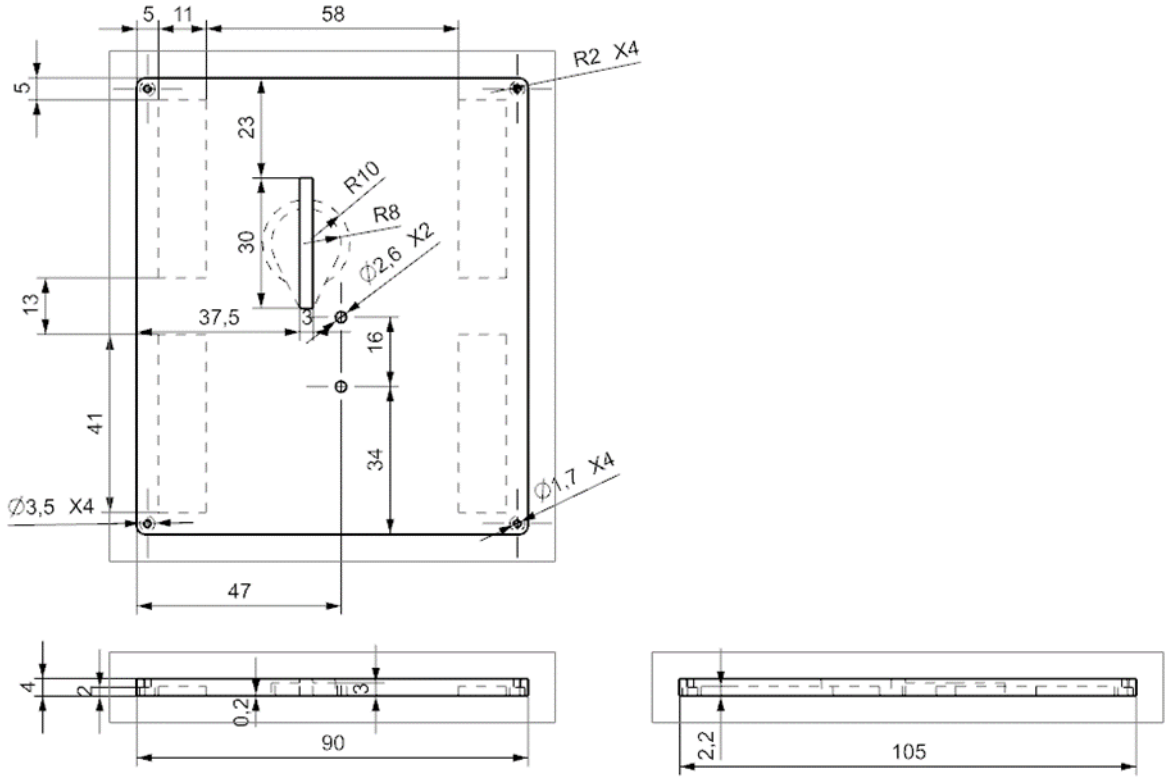
수집 장치는 라즈베리파이 보드와 마이크를 장착하는 뚜껑, 피에조 센서와 부착을 위한 자석을 장착하는 밑판으로 구성하였습니다. 뚜껑과 밑판은 나사를 사용하여 결합하였고 센서와 보드, ADC는 점퍼선과 납땜하여 회로를 연결하였습니다.

그림의 좌측이 뚜껑의 안쪽, 가운데가 밑판의 밑면, 우측이 조립 형태입니다.



상세수치는 하단에 뚜껑, 밑판 순서로 첨부하였습니다.




5
11
58
R2 X4
5
23
R10
R8
30
Ø2,6 X2
13
37,5
3
16
41
34
Ø3,5 X4
Ø1,7 X4
47
4
2
0,2
3
90
2,2
105

Data Pre-processing

Background Knowledge

두 센서를 통해 수집된 데이터의 원본은 시계열 데이터로 1열 3000행의 데이터 시트입니다. 1초에 3000개의 데이터를 수집하였으므로 열에 3000행의 데이터가 기록됩니다.

본 프로젝트에서는 정상 공구와 마모 공구를 이용해 수집된 데이터를 각각 Labeling하고 지도학습을 시킨 머신러닝 모델을 데이터 수집 모듈에 탑재시켜, 자율적인 공구 마모 진단이 가능케하는 것이 목적입니다.

따라서, 위에서 언급한 지도학습을 위해서는 원 데이터의 형식을 변환해야 할 필요가 있습니다. 1행에는 특정 상태의 공구에서 나온 데이터의 정보가 입력되고, 마지막 열에 정답열을 형성하여 해당 데이터가 어떤 상태의 공구를 이용한 것인지 답을 알려주는 방식을 사용할 예정입니다.

Raw-data는 3000행이므로, 이를 단순히 가로로 배열하여 3000개의 열 + 1열(정답열)을 형성하는 것은 무의미합니다. 즉, 전처리 과정을 거쳐 3000행의 데이터를 최대한 대표하는 특성을 남겨 압축시켜 약 6열 + 1열의 데이터시트를 생성하는 것을 목표로 해야 합니다. 아래에는 이 목표를 달성하기 위한 전처리 과정 4 steps을 정리해 두었습니다.

아래 기재된 내용과 예제 코드를 통해 전반적인 내용을 이해하시고, 상세 코드가 필요하시다면 최하단의 링크를 이용하시면 됩니다.

FFT & IFFT

앞선 정상 마모 공구 구분 실험에서부터 알 수 있었듯이, 센서의 품질이 그리 높지 않아 노이즈가 과하게 발생하는 것을 알 수 있었습니다. 그 중에서도 0Hz에서 magnitude가 매우 높게 수집되어 타 Hz에서의 값을 확인하는데 어려움이 있었습니다.

따라서 본 연구에서는 시계열 데이터를 주파수 도메인으로 변환한 뒤, 분석에 방해가 되는 Hz의 노이즈를 제거하고 다시 시계열 도메인으로 돌아오는 방식을 채택하였습니다.

이 과정에서는 데이터의 손실(변환 과정에서 허수부가 생성되며, 추후 전처리 및 지도학습 과정을 위해서는 이를 삭제해야 함)이 필연적으로 발생합니다. 그럼에도 불구하고 변환 및 재변환 과정을 거치는 이유는, 본 연구에서 다루는 신호인 진동과 소음은 비선형적 데이터일 뿐만 아니라, 단순히 주파수 성분으로 변환하여 분석할 경우 놓칠 수 있는 유의미한 패턴을 잡아내는 데 이점이 있기 때문입니다.

[코드 예시]

```
fft_result_piezo = np.fft.fft(time_series_data_piezo)

fft_result_mic = np.fft.fft(time_series_data_mic)
```

Standard Scaling

데이터 전처리 과정 중 하나로, 모든 데이터들을 정규 분포로써 스케일을 일치시켜, 서로의 특성을 비교하는데 이점을 가져가도록 하기 위함입니다.

[코드 예시]

```
scaler = StandardScaler()

scaled_real_n_p = scaler.fit_transform(real_part_n_p)
```

Statistical Features Extraction and Data Transformation

위에서 언급했다시피, 3000행을 그저 가로로 돌려놓는 것만으로는 아무 의미가 없습니다. 데이터를 대표하는 특성을 뽑아 분량을 확 줄이고, 그 몇개를 돌려 열로 만들어야 합니다. 본 연구에서는 그 특성을 통계적 특성으로 하였습니다.

앞서 Standard scaling을 하였으므로 평균과 표준편차, 분산은 제외한 특성들을 이용했습니다.

[코드 예시]

```
stat_piezo=pd.DataFrame({'min':sequence_piezo.min(numeric_only=True),

            'max':sequence_piezo.max(numeric_only=True),

            'quantile':sequence_piezo.quantile(numeric_only=True),

            'median':sequence_piezo.median(numeric_only=True),

            'skew':sequence_piezo.skew(numeric_only=True),

            'kurtosis':sequence_piezo.kurtosis(numeric_only=True)})
```

PCA

주성분 분석으로 불리는 차원축소법 중 하나입니다. 앞서 3번의 과정을 거치며 세로로 길던 데이터를 축약시켜 가로로 나열하게 되었습니다.

열은 데이터 분석에서 '차원'이라고도 불리우며, 이는 쉽게 생각하면 x축 같이 하나의 축이라고 볼 수 있습니다. 데이터를 미지의 공간에 펴놓고, 해당 데이터의 특성을 가장 잘 표현하는 선을 긋습니다. 그 선에 수직하면서 또 데이터의 특성을 가장 잘 표현하는 선을 긋습니다. 이 과정을 3번만 반복하면 원래 6차원의 데이터가 3차원으로 표현되며, 이걸 PCA라고 합니다.

공구 마모 진단 모듈 탑재용 코드에서는 2차원으로 줄였지만, 모델 지도학습에는 더이상의 데이터 손실을 막기 위해 6차원으로 하였습니다. '6차원 -> 6차원' 변환임에도 PCA를 통해 데이터의 특성을 가장 잘 표현하는 방식으로 변환되기에 분석적 이점이 있습니다.

[코드 예시]

```
from sklearn.decomposition import PCA

model_piezo_pca = PCA(n_components=6)

model_piezo_pca.fit(stat_piezo)
```

Answer Column Generation

각 행에는 서로 다른 가공의 데이터가 입력되어 있습니다.

이제 우측 마지막 열에 정답열을 추가하고 각 데이터가 어떤 상태의 공구로부터 나온 것인지 정답을 라벨링해줘야 합니다.

예를 들어, 1행의 데이터는 정상 공구의 데이터이니 NORMAL을, 2행의 데이터는 마모 공구의 데이터이니 WEAR를 입력해줍니다. ML모델은 이 ANSWER열을 기준으로 지도학습을 하게 됩니다.

Data Analysis and Machine Learning

Machine Learning Model Selection

머신러닝에 사용될 알고리즘 모델은 로지스틱 리그레션, 서포트 벡터 머신, 랜덤 프레스트, XG부스트로 Classification 계열의 알고리즘을 사용했습니다.

갖고 있는 데이터 중 75%는 모델의 학습에, 25%는 평가에 사용됩니다.

[코드 예시]

```
from sklearn.linear_model import LogisticRegression

model_lr_piezo = LogisticRegression()

model_lr_piezo.fit(X_train_piezo, y_train_piezo)

pred_lr_piezo = model_lr_piezo.predict(X_test_piezo)

from sklearn import metrics

from IPython.display import display, Markdown

display(Markdown('### Confusion Matrix'))

display(pd.crosstab(y_test_piezo, pred_lr_piezo, margins=True))

print(metrics.classification_report(y_test_piezo, pred_lr_piezo))

print('Accuracy: {}'.format(metrics.accuracy_score(y_test_piezo, pred_lr_piezo)))

print('Precision: {}'.format(metrics.precision_score(y_test_piezo, pred_lr_piezo, average='weighted')))

print('Recall: {}'.format(metrics.recall_score(y_test_piezo, pred_lr_piezo, average='weighted')))
```

각 모델의 평가 결과를 참고하여 최종적으로 사용할 알고리즘 모델을 선정합니다.

본 연구의 경우, 불필요한 공구 교체를 방지하고 적절한 시기에 유도함으로써 생산 효율을 늘리는 것이 목적이므로 Precision 값을 위주로 모델을 평가하였습니다. 그 결과 XGBoost가 가장 우수한 성적으로 모듈 탑재용으로 선정되었습니다.

학습된 머신러닝 모델은 진단 모듈에 탑재시키기 위해 pkl 형식으로 wrapping하여 모듈로 옮겨졌습니다. 모듈에 탑재된 라즈베리파이에서의 데이터 수집 코드는 지도학습용 데이터 수집 코드와 다를 것이 없습니다.

Design of the Final Diagnosis Result Extraction Logic

피에조 센서용 XGBoost와 마이크 센서용 XGBoost가 각각 존재하기에, 최종 진단 결과를 내려면 기준이 필요했습니다.

Classification의 결과 도출 원리는, threshold를 기준으로 그보다 높으면 1, 낮으면 0으로 처리하는 방식입니다. 즉, 0과 1이 도출되기 전에 원 값이 존재하며, 이를 이용하면 됩니다.

이 원 값이 threshold에서 멀리 떨어져 있을 수록, 즉 0 혹은 1에 가까이 있을 수록 보다 확실하게 0 혹은 1을 주장하는 것을 의미합니다. 따라서, 피에조 센서와 마이크 센서가 각각 공구의 상태를 다르게 진단했을 때, 이 원 값을 비교하여, threshold로부터 더 멀리 떨어진 값을 가진 센서의 알고리즘의 결과값을 따르는 것이 합리적입니다.

[코드 예시]

```
for index, row in result_xgb.iterrows():
  if row['PIEZO_XGB_PREDICT'] == row['MIC_XGB_PREDICT']:
      if row['PIEZO_XGB_PREDICT'] > 0.5:
         result_xgb.at[index,'RESULT'] = 'WEAR'
      else:
         result_xgb.at[index, 'RESULT'] = 'NORMAL'
  else:
      piezo_diff = abs(row['PIEZO_XGB_PREDICT'] - 0.5)
      mic_diff = abs(row['MIC_XGB_PREDICT'] - 0.5)

      if piezo_diff > mic_diff:
         if row['PIEZO_XGB_PREDICT'] > 0.5:
            result_xgb.at[index, 'RESULT'] = 'WEAR'
         else:
            result_xgb.at[index, 'RESULT'] = 'NORMAL'
      else:
         if row['MIC_XGB_PREDICT'] > 0.5:
            result_xgb.at[index, 'RESULT'] = 'WEAR'
         else:
            result_xgb.at[index, 'RESULT'] = 'NORMAL'
```

Module Installation and Detailed Code

지도학습용 데이터 수집과 지도학습, 진단 모듈 탑재용 상세 코드는 이곳에서 branch를 열어 확인하시면 됩니다.

The Overall Flow and Report of the Research

전반적인 연구의 흐름이 담긴 최종레포트는 위 상세 코드 확인을 위한 링크에서 다운로드 받으실 수 있습니다.

감사합니다.