2023-2 파란학기제 최종 성과물

Development of Deep learning – based Planetary gear condition Diagnosis technology

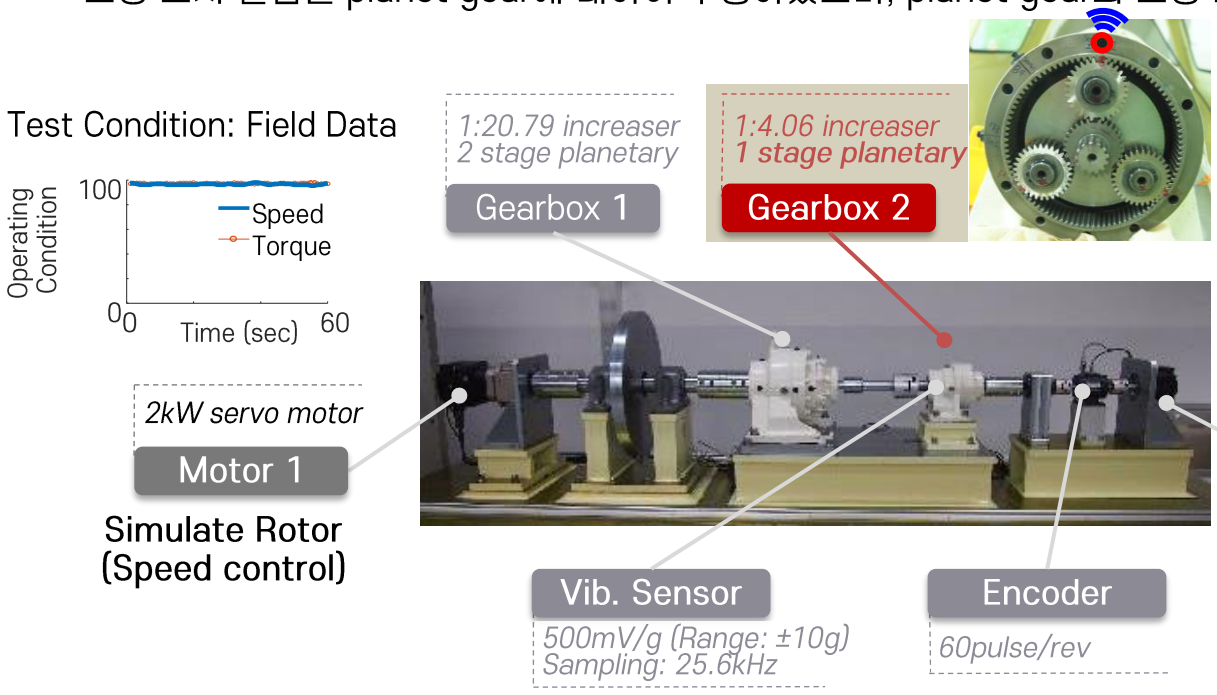
딥러닝 기반 유성기어 상태진단 기술 개발

<TopGEAR> 김인태, 문세희

발표자 : 김인태, 문세희
소속 : 아주대학교 산업공학과
이메일 : kitkit8142@ajou.ac.kr, anstpgml5035@ajou.ac.kr

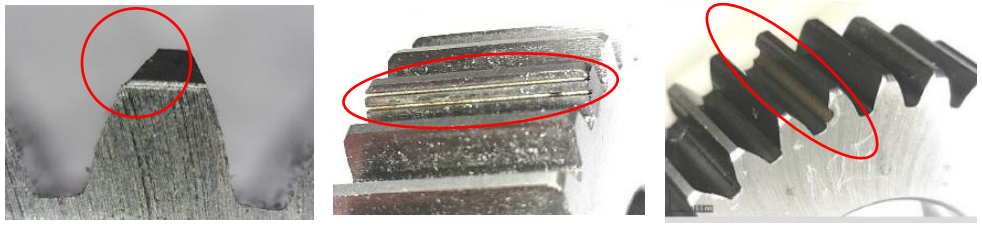
01. Testbed Setup

- 데이터는 한국표준과학연구원과 서울대학교에서 공동으로 실험한 데이터를 사용하였으며, Testbed의 구성은 다음과 같음.
- 고장 모사 실험은 planet gear에 대하여 수행하였으며, planet gear의 고장 모사는 다음과 같음.



Gear	# of Teeth
Planet	31
Ring	95
Sun	31

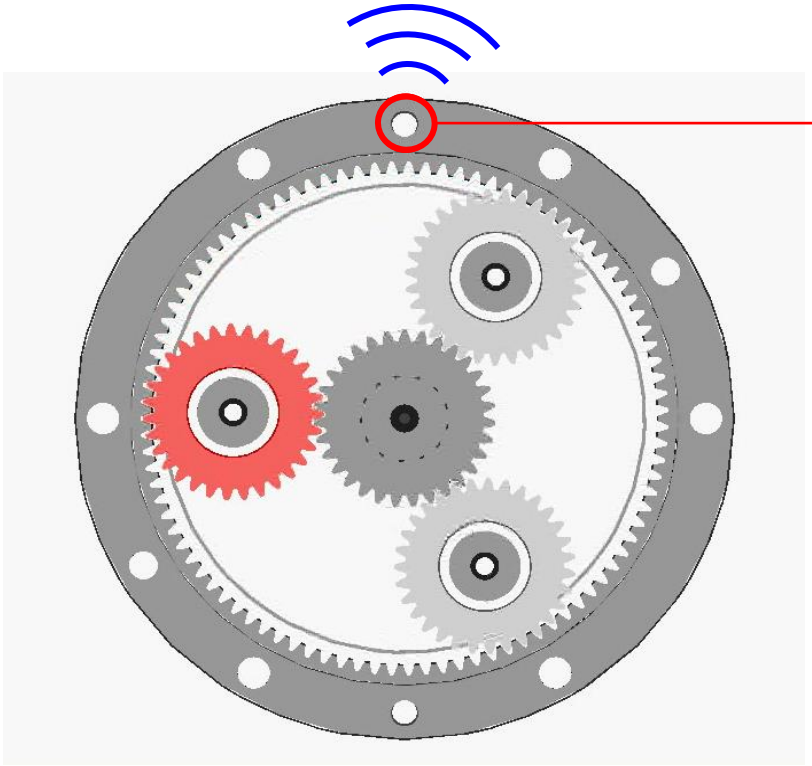
<Fig.1> 한국표준과학연구원 및 서울대학교에서 공동으로 수행한 고장 모사 실험의 Testbed

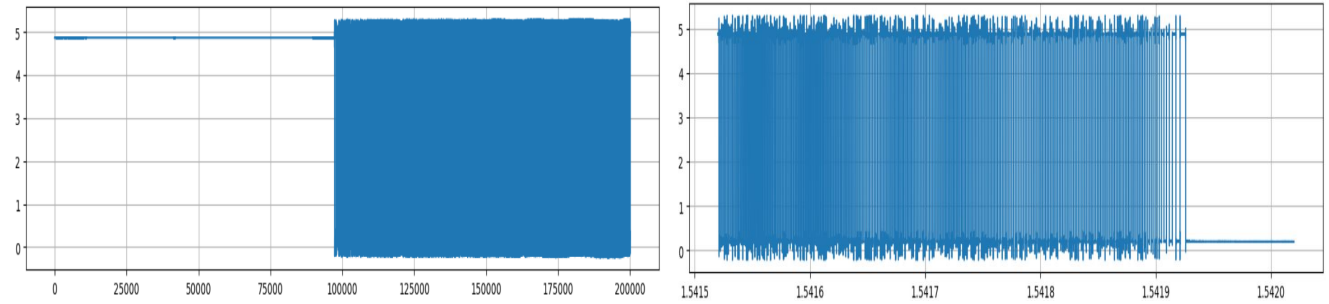


<Fig.2> 실험에 사용된 fault of planet gears

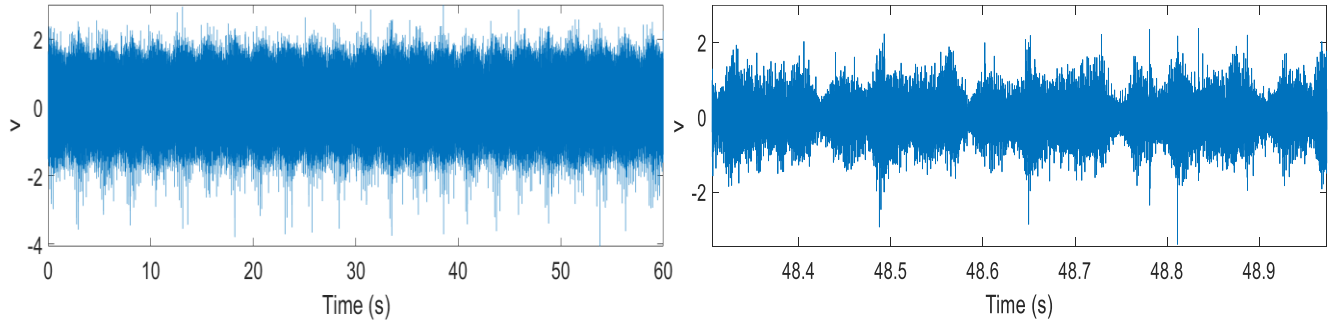
02. Data Preprocessing

- Sampling frequency: 25600Hz
- 고장 모사 실험은 총 100min을 진행 => 취득된 normal/fault signal: 154,202,000 sample





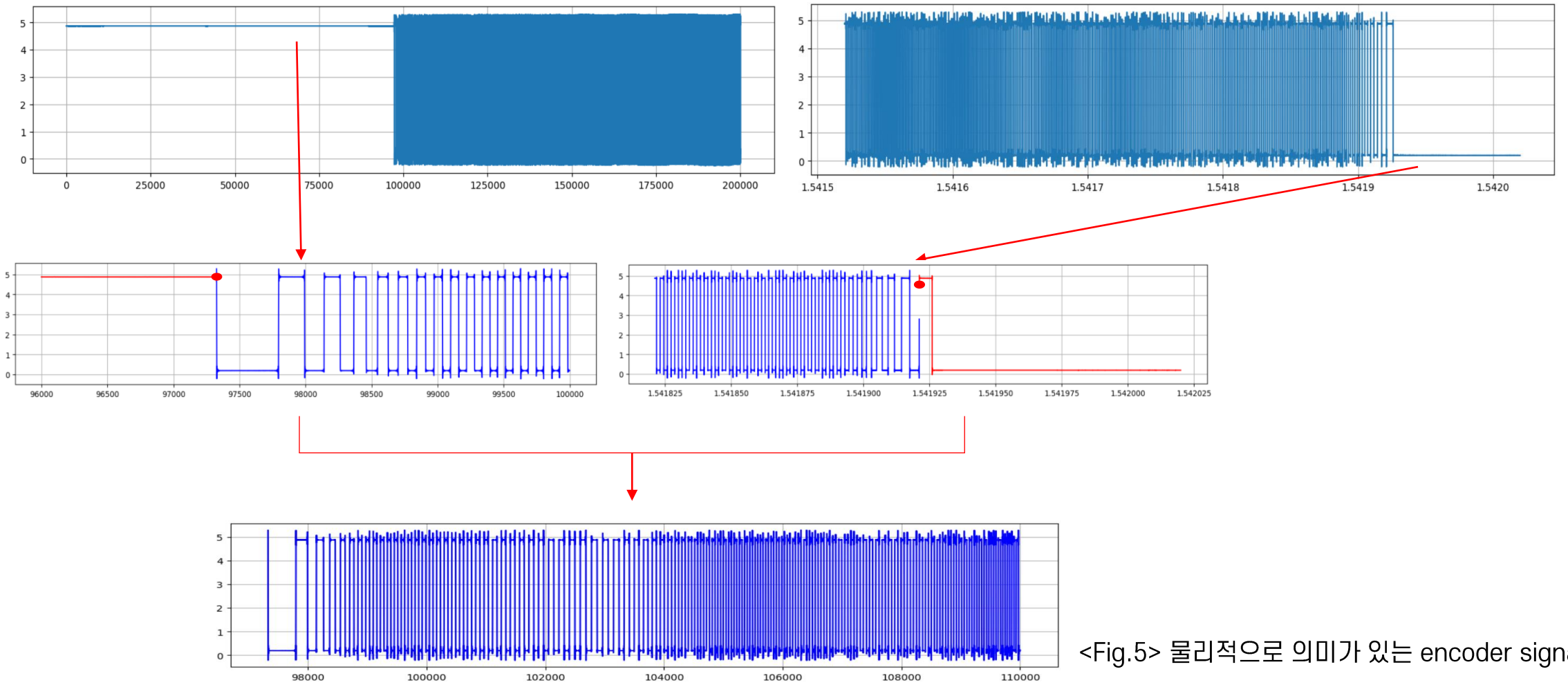
<Fig.3> 가속 센서로부터 계측한 encoder signal의 앞/뒤



<Fig.4> 가속 센서로부터 계측한 vibration signal

02. Data Preprocessing

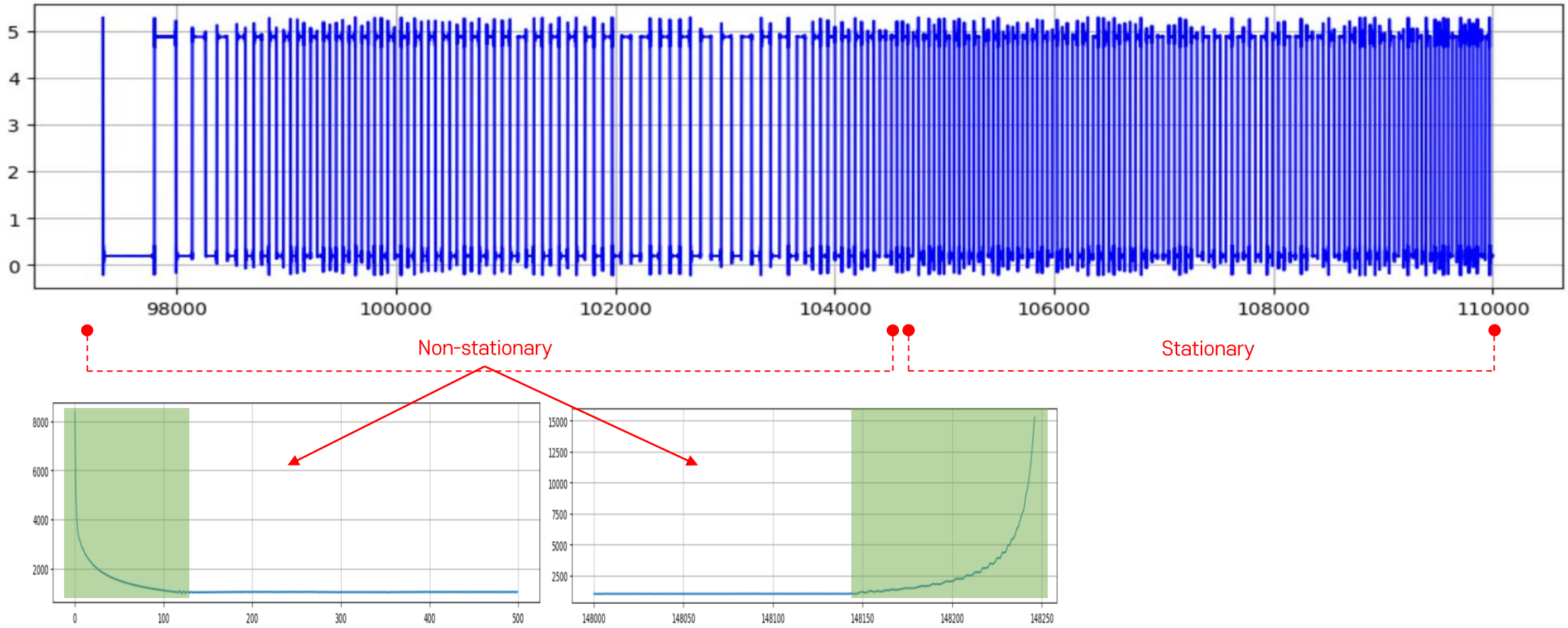
- 취득된 encoder데이터의 표시한 부분은 물리적으로 의미가 없다고 판단 => 제거



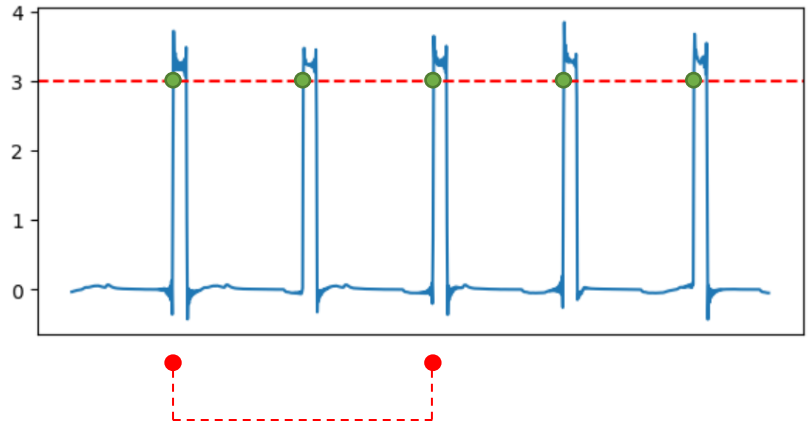
<Fig.5> 물리적으로 의미가 있는 encoder signal

02. Data Preprocessing

- Encoder signal을 살펴봤을 때, Stationary하지 않은 구간이 존재 => pulse/rotation을 만족시켜야 함.



```python
class Angular_Resampling():
    def __init__(self, data_encoder, data_vibration, fs, ppr):
        self.data_encoder= data_encoder
        self.data_vibration= data_vibration
        self.fs= fs
        self.ppr= ppr

    def resampling(self, num_divisions):
        self.num_divisions= num_divisions
        index_list= list()

        flag= False
        for idx, val in enumerate(self.data_encoder):
            if not flag and val> 3:
                flag = True
                index_list.append(idx)

            elif flag and val< 3:
                flag = False

        index_list= index_list[self.ppr - 1:: self.ppr]

        def extract_start_end(angles):
            #angles 리스트에서 시작 각도와 끝 각도를 추출
            start_end_pairs= [(angles[i], angles[i + 1]) for i in range(len(angles) - 1)]

            return start_end_pairs

        def resample_angles(start, end, num_samples):
            #주어진 시작 각도와 끝 각도 사이에서 등간격으로 리샘플링하는 함수
            sampled_angles= np.linspace(start, end, num_samples)

            return sampled_angles

        angles= [(i * 360) for i in range(len(index_list))]

        start_end_pairs= extract_start_end(angles)

        resampled_angles= list()
        for start, end in start_end_pairs:
            resampled_interval= resample_angles(start, end, self.num_divisions)
            resampled_angles.extend(resampled_interval[:-1])  #same interval

        inverse_cs= PchipInterpolator(angles, index_list) #각함수 생성
        resampled_time= inverse_cs(resampled_angles) #등간격 각->시간 추출

        cubic_interp1= CubicSpline(self.data_vibration.index, self.data_vibration)

        resampled_data= cubic_interp1(resampled_time)

        resampled_data= pd.DataFrame({'Channel_1': resampled_data})

        return resampled_data
```

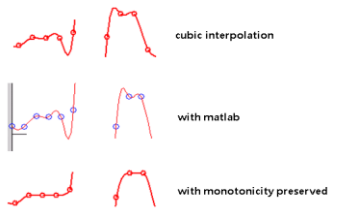


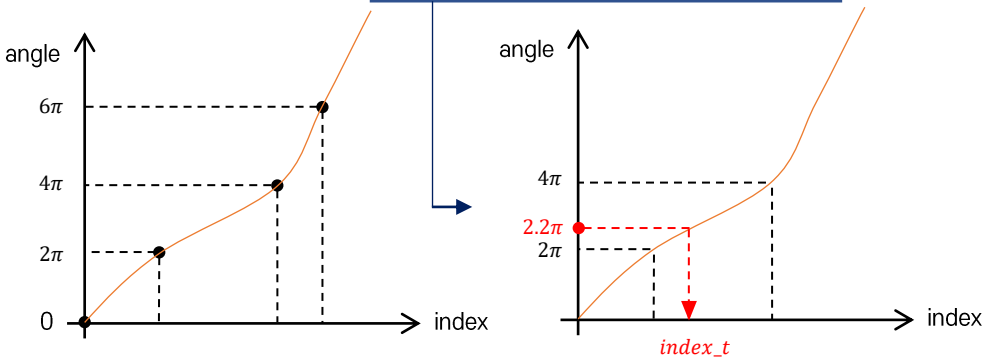
ppr (pulse per revolution) = 2π

$[[0, 2\pi], [2\pi, 4\pi], [4\pi, 6\pi] \cdots]$

if ppr== 10

$[\cdots [2\pi, 2.1\pi, 2.2\pi, \ldots, 4\pi] \cdots]$



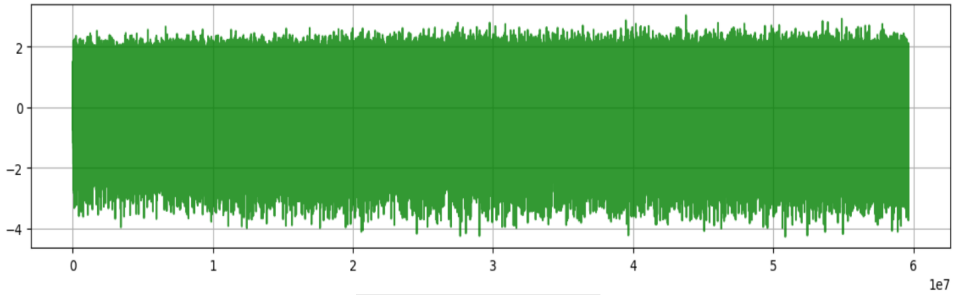


<Fig.6> pulse/rotation을 만족시키기 위한 Angular-based Resampling

03. Autocorrelation-based Window function

- Angular-based Resampling을 적용한 결과, rotation당 일정한 sample을 가짐: 59,650,851 sample
- Time Series에서 continuous/discrete signal이 자기 자신과 얼마나 유사한지 유사도를 측정하는 “Autocorrelation function”의 성질을 이용

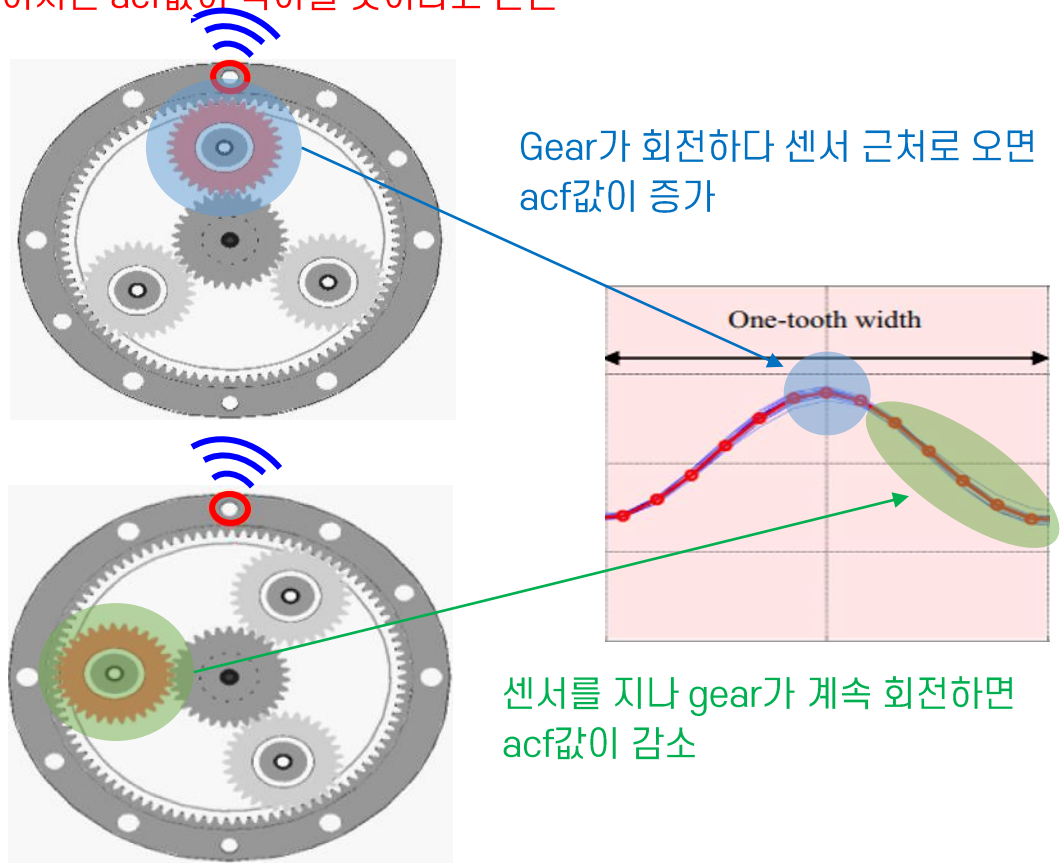
=> Sensor아래에 고장 기어가 위치했을 때, acf값이 커질 것이고, sensor와 멀어지면 acf값이 작아질 것이라고 판단



	Channel_1
0	-0.025993
1	-0.158967
2	0.149580
3	-0.122686
4	-0.095593
...	...
59650846	-0.444170
59650847	-0.551364
59650848	0.336324
59650849	-0.275383
59650850	-0.210905

59650851 rows × 1 columns

<Fig.7> Angular-based Resampling을 수행한 vibration signal



<Fig.8> fault gear의 위치와 autocorrelation function value의 관계

03. Autocorrelation-based Window function

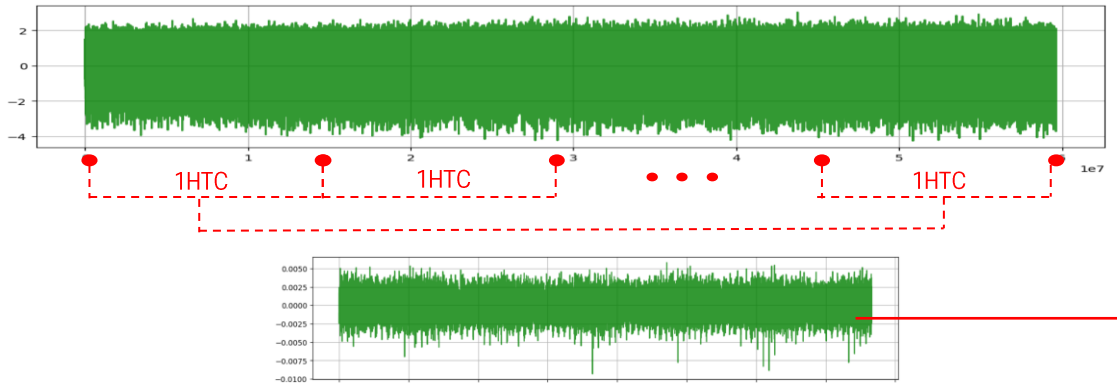
- 아래의 Autocorrelation function을 정의하는 수식을 활용하여 autocorrelation function 함수를 구현.

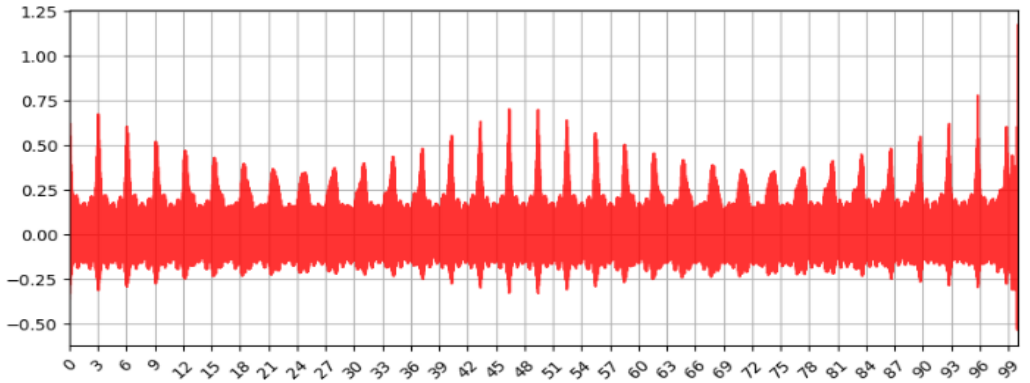
$$R_{vv}(\tau) = E[v_{rs}(t)v_{rs}(t+\tau)]$$

- Gear가 회전하다가 초기의 상태로 돌아오는 1HTC를 기준으로 autocorrelation function 함수를 적용
- Autocorrelation function value를 planet gear rotation domain으로 변환
- autocorrelation function 함수를 적용한 값들을 평균 => noise제거를 통해 결과를 더 명확하게 볼 수 있게 함

```
#acf함수 구현
def acf(data, t_lag):
    data= np.array(data).reshape(-1)
    mean= data.mean()

    numerator= np.sum((data[: len(data) - t_lag] - mean) * (data[t_lag:] - mean))
    denominator= np.sum(np.square(data - mean))

    acf_val= numerator / denominator

    return acf_val
```

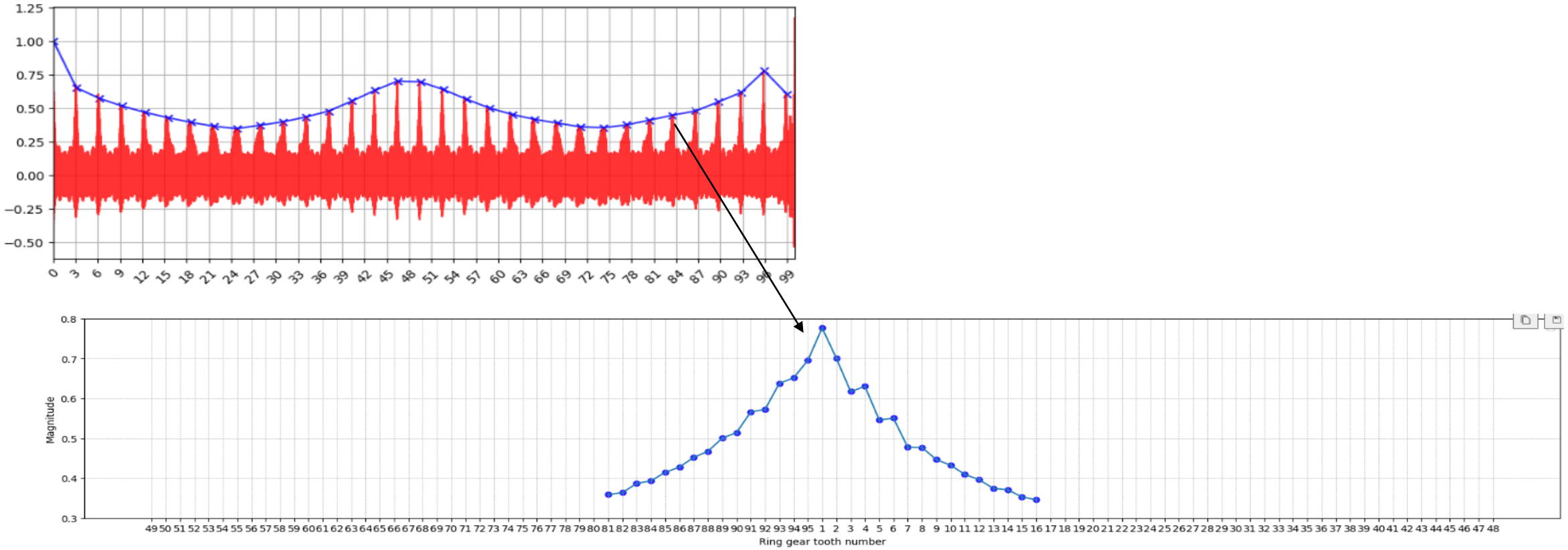
$R_{vv}(\tau) = E[v_{rs}(t)v_{rs}(t+\tau)]$





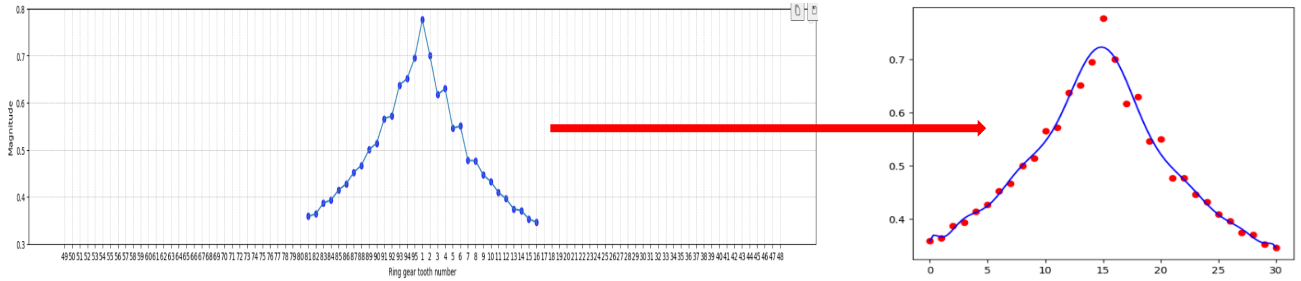
03. Autocorrelation-based Window function

- Autocorrelation function value의 local maximum값들을 추출 해야함 => Hilbert transform을 통한 envelop
- ring gear domain으로 변환
- Autocorrelation function을 통해 fault gear가 sensor아래에 위치할 때, 값이 커지게 됨
- 위 결과를 window function으로 활용해 raw signal 합성하면 sensor아래에 fault gear가 존재할 때 signal이 증폭되고, 멀어질 때 0으로 바뀜

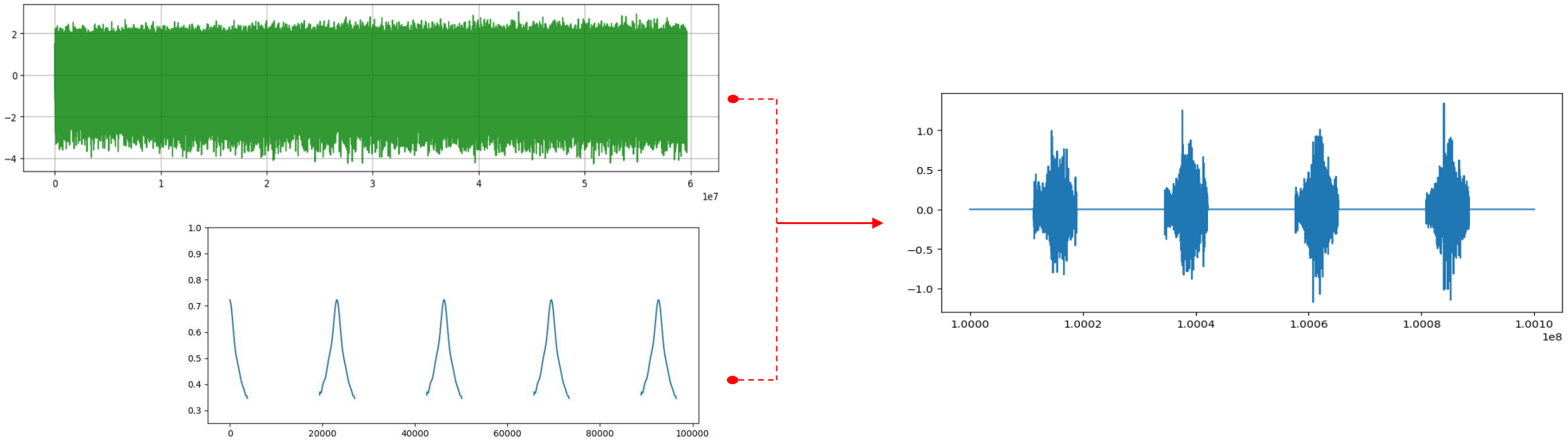


04. Model Implementation

- Autocorrelation function의 local maximum값들에 대한 envelope신호를 window function함수로 추출

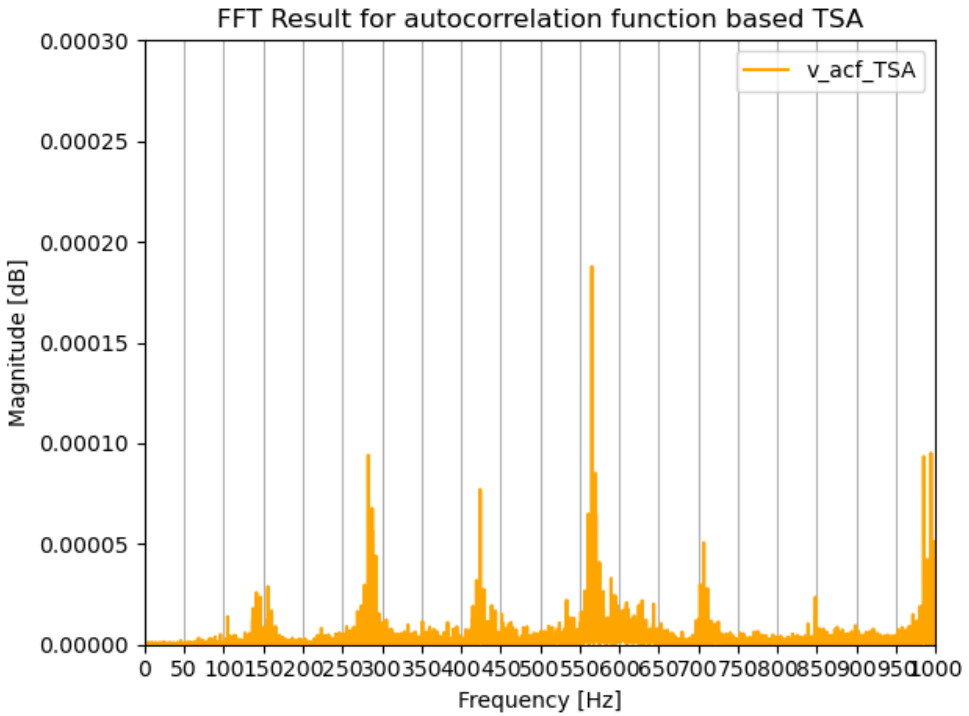


- Resampling된 signal에 autocorrelation-based window function을 합성한 뒤, Time Synchronous Averaging기법을 적용

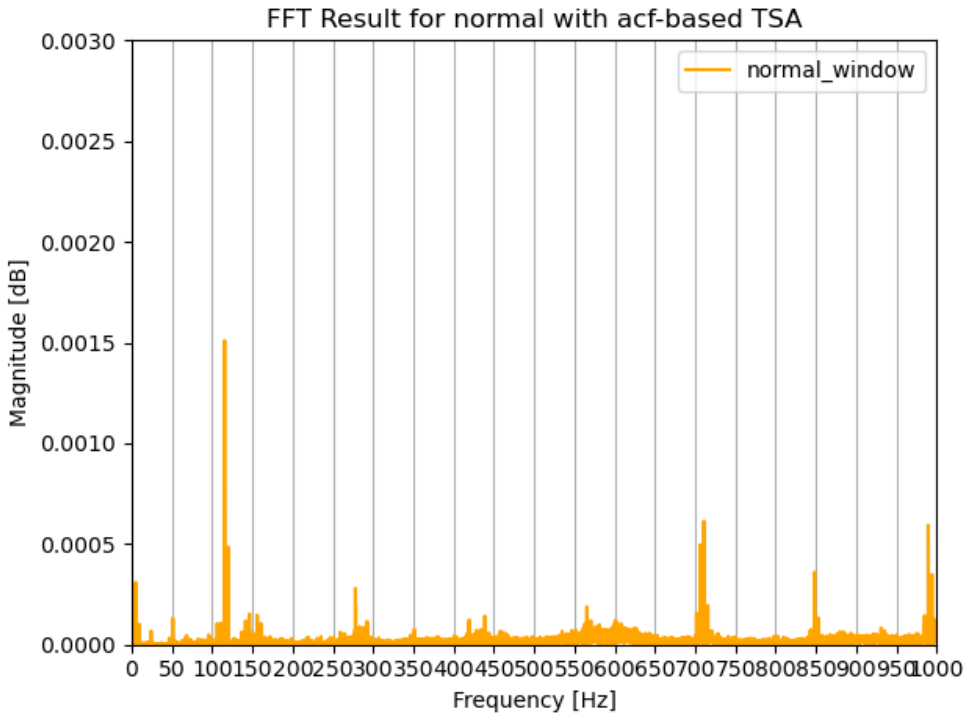


04. Model Implementation

- 앞서 개발한 ACF - based TSA를 Normal, Fault 데이터에 적용시킨 후, 주파수 영역으로 변환해 확인해 본 결과는 다음과 같음.



<Fig.9> fault data에 대하여 fft를 수행한 결과



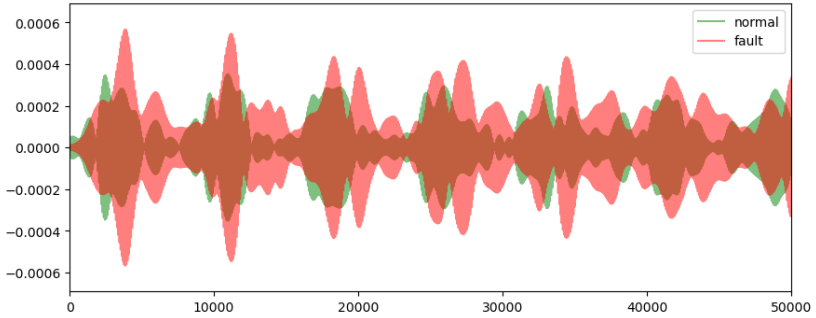
<Fig.10> normal data에 대하여 fft를 수행한 결과

04. Model Implementation

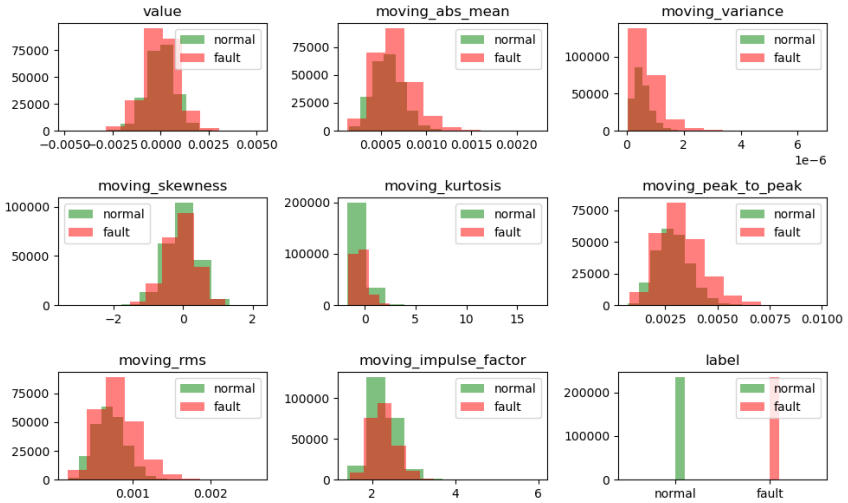
- Window function을 적용한 normal, fault데이터에 대해 주파수 영역에 차이가 존재하는 550hz ~ 600hz 영역에 band pass filter를 적용함
- 더 유의미한 성분을 도출하기 위해 아래와 같은 파생변수를 통해 파생변수화를 진행

특징	공식	기대효과
Absolute mean	$\frac{1}{n}\sum_{i=1}^{n}\lvert x_i-\bar{x}\rvert^2$	진동 신호의 전반적인 수준의 변화를 감지
Variance	$\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2$	진동 신호의 확산 정도 감지
Skewness	$\frac{1}{n}\frac{\sum_{i=1}^{n}x_i^3}{\sigma^3}$	진동 신호의 비대칭성 감지
Kurtosis	$\frac{1}{n}\frac{\sum_{i=1}^{n}x_i^4}{\sigma^4}$	진동 신호의 평탄성 감지
Peak-to-Peak	$x_{\max}-x_{\min}$	진동 진폭의 전체 범위를 측정
RMS	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}$	진동 신호의 평균 에너지 측정
Impulse factor	$\frac{\max(\lvert x_i\rvert)}{\frac{1}{n}\sum_{i=1}^{n}\lvert x_i\rvert^2}$	진동 신호의 고주파 함량 측정

<Fig.11> 7가지 파생변수를 통한 파생변수화 진행



<Fig.12> band pass filter를 적용한 진동신호 시각화



<Fig.13> 상태별 파생변수 히스토그램

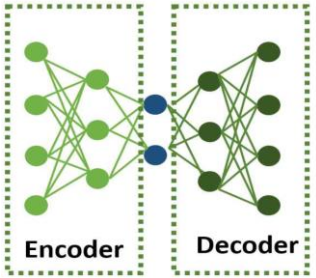
04. Model Implementation

- 대표적인 비지도학습 모델인 AutoEncoder를 활용한 이상 탐지 모델 구축
- 오토인코더 모델을 정상 데이터의 패턴을 학습시킨 뒤, 이상 데이터를 입력하여 재구성 오차를 MSE(Mean Squared Error)를 활용하여 계산
- 미리 설정한 임계값(Threshold)와 비교하여 임계값 이상의 재구성 오차를 가진다면 이상 데이터라고 판별함.

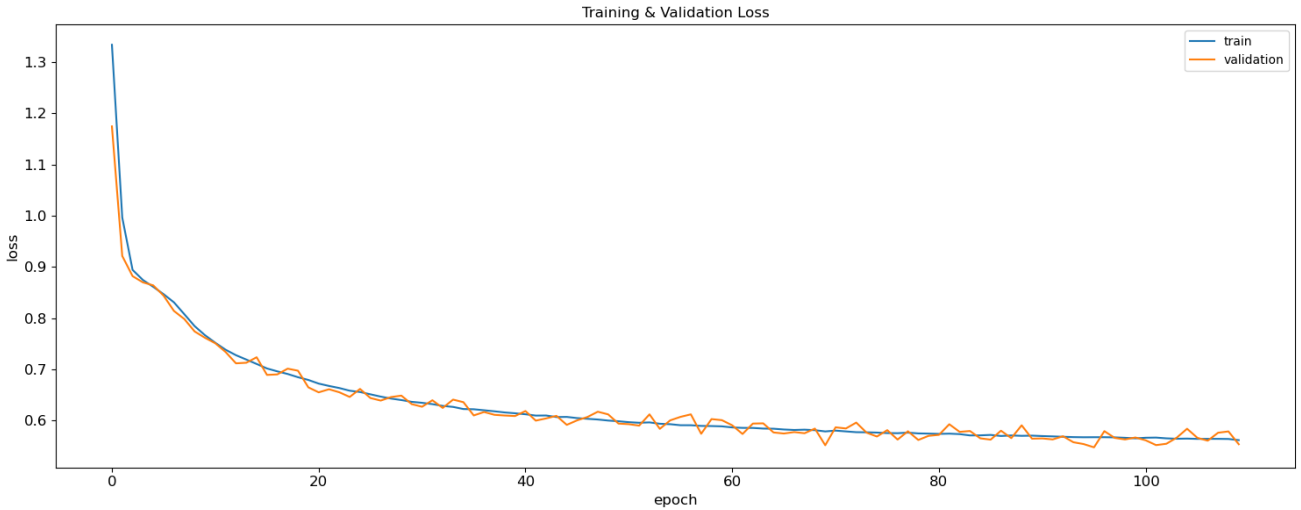
```python
# 오토인코더 모델 정의
class Autoencoder(nn.Module):
    def __init__(self, input_size):
        super(Autoencoder, self).__init__()
        # 인코더
        self.encoder = nn.Sequential(
            nn.Linear(input_size, 8),
            nn.ReLU(True),
            nn.Linear(8, 16),
            nn.ReLU(True),
            nn.Linear(16, 8),
            nn.ReLU(True),
            nn.Linear(8, 4),
            nn.ReLU(True)
        )
        # 디코더
        self.decoder = nn.Sequential(
            nn.Linear(4, 8),
            nn.ReLU(True),
            nn.Linear(8, 16),
            nn.ReLU(True),
            nn.Linear(16, 8),
            nn.ReLU(True),
            nn.Linear(8, 4),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x

# 데이터셋 및 데이터 로더 초기화
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
dataset = DataFrameDataset(normal)
dataloader = DataLoader(dataset, batch_size=128, shuffle=False)

# 모델 초기화
input_size = normal.shape[1]
model = Autoencoder(input_size).to(device)

# 손실 함수 및 최적화기
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

활용한 모델의 구조는
- 인코더 4개층
- 디코더 4개층
- 잠재 백터 1개층

총 9개의 레이어로 구성하였으며, 은닉층은 모두 완전 연결 층으로 이루어져 있음. 활성화 함수로는 모드 ReLU 함수를 사용함.



<Fig.14> Autoencoder 구조 예시



<Fig.15> epoch 학습에 따른 loss 시각화

05. Result

본 프로젝트에서 활용한 AutoEncoder 모델을 RandomForest Classification 모델을 대조군으로 하여 정확도를 평가해 본 결과는 다음과 같음.

Classification Report:

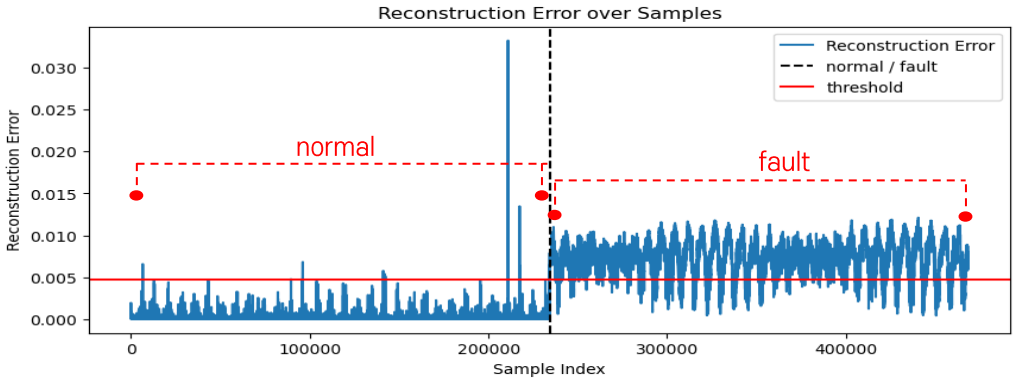
	precision	recall	f1-score	support
0	0.89	0.92	0.91	46913
1	0.92	0.89	0.90	46794
accuracy			0.90	93707
macro avg	0.91	0.90	0.90	93707
weighted avg	0.91	0.90	0.90	93707

<Fig.16> RandomForest Classification을 활용한 결과

Classification Report:

	precision	recall	f1-score	support
False	0.88	1.00	0.93	234266
True	1.00	0.86	0.92	234266
accuracy			0.93	468532
macro avg	0.94	0.93	0.93	468532
weighted avg	0.94	0.93	0.93	468532

<Fig.17> Autoencoder를 활용한 결과



<Fig.18> 재구성 오차의 57%를 임계점으로 설정하고 이를 넘었을 경우, 이상으로 분류하는 Autoencoder 모델

- 위 결과에서 볼 수 있듯, 머신러닝 기법을 활용하여 이상감지를 수행한 것에 비해 딥러닝 방법을 사용하여 이상감지를 수행한 것이 더 높은 정확도를 보이는 것을 알 수 있음.
- 또한, 이상 데이터의 레이블이 없거나 드물게 발생하는 실제 현장에서, 비지도학습 기반의 AutoEncoder 모델은 큰 강점을 가지고 있음.

Thank you

김인태, 문세희
아주대학교 산업공학과
kitkit8142@ajou.ac.kr, anstpgml5035@ajou.ac.kr