Kuwait University

COLLEGE OF SCIENCE

PROGRAMMING PARADIGMS

TIME SERIES FEATURES EXTRACTION ALGORITHMS

Spring 2024/2025

Student Name	Student ID
Abeer Alsafran	224125356

Submission Date : 10/05/2025

Contents

List of Figures

1 Introduction

Feature extraction has so many purposes, it can be used for Machine Learning and AI, Computer Vision, Natural Language Processing (NLP), Medical Imaging, Audio and Speech Recognition, Finance, Robotics, and Biometrics. In this project, we focus on the ML usage in time series feature extraction. Not only does it help improve the model's accuracy, but it also adds semantic meaning to computer vision tasks.
In this work, We are interested to test the **TSFRESH** library on the UAV sensors **indoor2 Upward spiral** dataset on parallel paradigm. We will run the algorithm in a an environment which doesn't have the parallel paradigm, and on another environment that have the parallel paradigm, and then we will compare the difference and check the performance matrix how does it perform.
We can expect that the parallel paradigm would be much better in matrix performance. And that's because it will decrease the latency and increase the speed of processing. With that being said, it is beneficial to explore what other researcher have done so far this will be discussed in section 2. Then we will discuss in section 3 the semantics and the syntax of the chosen programming language. Section 4 will be explaining the methodology after that we will check the result of the performance matrix.

2 Related Work

There are a lot of researchers who have done the same in this area. One of them is [1] A review of UAV flight data anomaly detection. It gives a review about the flight data and how it impact the safety of the UAV. In paper [2] they investigate an unmanned aerial vehicle (UAV)-assisted Internet-of-Things (IoT) system in a sophisticated three-dimensional (3D) environment, where the UAV's trajectory is optimized to efficiently collect data from multiple IoT ground nodes. They have created a DRL based algorithm, TD3- TDCTM, which can design an efficient flight trajectory for a UAV to collect data from IoT nodes in a practical 3D urban environment with imperfect CSI. [3] study proposes an IoT UAV swarm anomaly detection model based on multi-modal DAEs and federated learning. First, according to the characteristics of UAV sensor data, a new loss function is designed by introducing the NCC into the MSE loss function. NMSE allows the model to better fit the clean data. Second, a multi-modal DAE is designed to extract multi-modal features using heterogeneous neural networks to improve the denoising capability of the model; the boldfaced squares indicate the vectors that the current model pays increasing attention to, such as the attitude category (A) vector focused on by the MLP in the data. Finally, the L-MDAE model is pre-trained with noise data, and the trained parameters are frozen to form a denoising model. In general, most papers relies on the extracted features from sensors data to perform critical tasks, such as trajectory prediction and UAV safety.

3 Semantics and Syntax

3.1 Syntax

Python is a high-level, interpreted programming language known for its clear and readable syntax. Its syntax is designed to be intuitive, reducing the complexity typically associated with other languages.

1- **Indentation:** Python uses indentation (whitespace) to define blocks of code rather than braces or other delimiters. Proper indentation is mandatory, as Python uses it to determine the grouping of statements.

2- **Variables and Data Types:** Variable types are dynamically assigned when a value is assigned to a variable. This means there is no need to explicitly declare the data type. For example:

```
is_active = True
```

3- **Comments:** Single-line comments are created by placing a hash symbol # before the comment text.

4- **Control Flow:** Provides several ways to control the flow of execution in a program, including conditional statements (if, else, elif) and loops (for, while).

5- **Functions:** Functions are defined in Python using the def keyword, and they allow code to be modularized and reused.

6- **Data Structures:** Offers several built-in data structures for storing collections of data, including lists, tuples, dictionaries, and sets.

7- **Error Handling:** A mechanism for handling runtime errors using try, except, and finally blocks. For example:

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("You can't divide by zero!")
finally:
    print("This will always execute."
```

8- **Object-Oriented Programming (OOP):** class has an __init__ method (constructor) to initialize the object's attributes.

9- **Importing Modules:** Python allows the use of external libraries and modules to extend functionality. These modules can be imported using the import keyword. For example:

```
import pandas as pd
```

3.2 Semantics

The semantics of a programming language refer to the meaning of its constructs, i.e., how the code behaves when executed. While the syntax of a language defines the structure and rules for writing code, the semantics define the rules for how expressions, statements, and other constructs execute and interact with one another. This report explores the semantics of the Python programming language and explains how Python supports different programming paradigms, including imperative, object-oriented, and functional programming.Python is a high-level programming language that provides a rich set of constructs for expressing algorithms and solving problems. Understanding Python's semantics is crucial to fully grasping how the language behaves during execution. This section outlines how Python's semantics support different programming paradigms.

3.2.1 Imperative Programming Paradigm

In the imperative programming paradigm, programs are defined as a sequence of statements that change the state of the system step-by-step. Python, being a general-purpose language, allows for imperative programming by providing constructs for variable assignment, conditionals, and loops.

1- **Variable Binding and Assignment:** Variables are references to objects in memory. When a value is assigned to a variable, the variable is bound to that object. The semantics of Python assignments involve referencing objects, and the assignment creates a mapping between a variable and a specific object in memory. For example:

```
x = 3
```

2- **Control Flow:** Python supports traditional imperative control structures such as if, else, for, and while. The semantics of these control structures dictate how Python evaluates conditions and how execution flows from one statement to another. For example:

```
for i in range(5):
    print(i)
```

3.2.2 Object-Oriented Programming (OOP) Paradigm

1- **Classes and Objects:**The semantics of object instantiation involve associating an instance of a class with a unique memory location. Each object can have methods (functions) and attributes (data fields) that define its behavior and state. For example:

```
class Person:
    def __init__(self, name):
        self.name = name

    def greet(self):
        print(f"Hello, my name is {self.name}")

p = Person("Alice")
p.greet()
```

2- **Inheritance and Polymorphism:** New classes to inherit the attributes and methods of existing classes. The semantics of inheritance define how a subclass inherits behavior from its superclass and how this behavior can be overridden or extended. For example:

```
class Animal:
    def speak(self):
        print("Animal sound")

class Dog(Animal):
    def speak(self):
        print("Woof")

d = Dog()
d.speak()
```

4 Dataset

The INSANE Dataset - Cross-Domain UAV Data Sets with Increased Number of Sensors for developing Advanced and Novel Estimators dataset is open source

Number	Column Name	Column Type
1	t	Time
2	a_x	Acceleration X
3	a_y	Acceleration Y
4	a_z	Acceleration Z
5	w_x	Velocity X
6	w_y	Velocity Y
7	w_z	Velocity Z

Table 1: Dataset columns type

[5] and it has multiple types of UAV Sensors datasets such as indoor or outdoor. In this project, we are extracting the features from indoor motion capturing. The dataset captures an upward spiral trajectory covering a distance of 25 meters, with a maximum height of 5 meters and a maximum velocity of 0.3 m/s, and the calibration is KLU1. Specifically, RealSense IMU format with Measurements: 35645,Sensor rate: 200.26Hz. In table 1, you can see the dataset columns type.

5 Methodology

5.1 Pre-Processing

In the Pre-Processing phase, we explore the data, visualize it, and clean it. We can see in Figure 1 that at time 180 there has been some up-normal behavior and from time 20 to 35 the status was steady. Figure 2 shows the overall appearance of the data set.

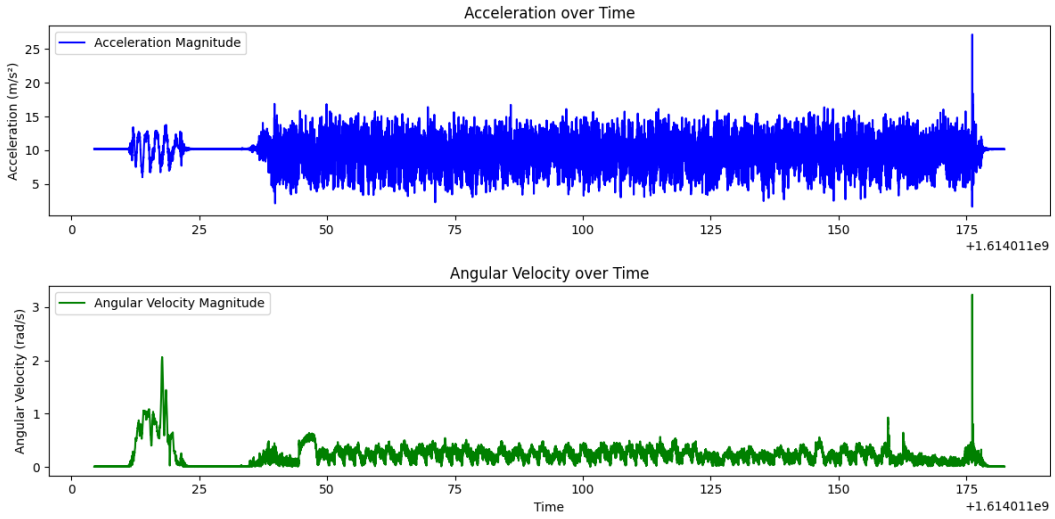

Figure 1: Magnitude Acceleration and Velocity over Time

	Data Type	Missing Values%	Unique Values%	Minimum Value	Maximum Value	DQ Issue
t	float64	0.000000	NA	1614011004.473068	1614011182.462577	No issue
a_x	float64	0.000000	NA	-9.933211	9.154444	Column has 2504 outliers greater than upper bound (3.65) or lower than lower bound(-3.76). Cap them or remove them.
a_y	float64	0.000000	NA	-3.507114	22.698407	Column has 430 outliers greater than upper bound (13.29) or lower than lower bound(-0.09). Cap them or remove them.
a_z	float64	0.000000	NA	-14.881703	1.435488	Column has 1685 outliers greater than upper bound (-2.25) or lower than lower bound(-8.76). Cap them or remove them.
w_x	float64	0.000000	NA	-0.744643	3.217197	Column has 1637 outliers greater than upper bound (0.28) or lower than lower bound(-0.29). Cap them or remove them.
w_y	float64	0.000000	NA	-2.176402	2.521558	Column has 1124 outliers greater than upper bound (0.41) or lower than lower bound(-0.25). Cap them or remove them.
w_z	float64	0.000000	NA	-0.757426	1.507395	Column has 1250 outliers greater than upper bound (0.21) or lower than lower bound(-0.34). Cap them or remove them.

Figure 2: Over all Dataset insights

```
(venv) abeer@Abeers-Air home % python3 main_v2.py
Extracting ....

Feature Extraction: 100%|| 25/25 [24:35<00:00, 59.03s/it]
/Users/abeer/home/venv/lib/python3.13/site-packages/tsfresh/utilities/dataframe_functions.py:198: RuntimeWarning: The columns [' w_y
__mean_abs_change' ' w_y__mean_change'
 ' w_y__mean_second_derivative_central' ...
 ' w_x__permutation_entropy__dimension_7__tau_1'
 ' w_x__query_similarity_count__query_None__threshold_0.0'
 ' w_x__mean_n_absolute_max__number_of_maxima_7'] did not have any finite values. Filling with zeros.
  warnings.warn(
Feature extraction time: 1769.88 seconds
```

Figure 3: Sample Output (Sequential)

5.2 Processing

In this phase, we implement our algorithm to execute the feature extraction on time series dataset. After exploring the data, we can identify the features that can be extracted. In our case, we have implemented the feature extraction function from tsfresh library.[5]

5.2.1 Sequential

First, it reads a time series dataset from a CSV file (dataset/rsimu.csv), assuming each row is timestamped data. Second, it adds a unique id to each row (for TSFRESH grouping). Next, extracts time-series features using the tsfresh library, which automates the generation of statistical features based on the id and timestamp t columns. And then imputes missing values in the extracted features. Finally, it clusters the extracted features using KMeans into 3 clusters after normalizing them. Reduces dimensionality with PCA for visualization and plots the clustered data in 2D. For more isights, it logs the time taken for feature extraction into a file (timediff.txt). Figure 3 shows the output for the sequential run. For more details about the Sequential run, we have used the code in **appendix A**.

5.2.2 Parallel

This script performs time series feature extraction, clustering, and visualization similarly to the sequential version, but it leverages parallel processing to improve performance. Using tsfresh and a MultiprocessingDistributor with 10

```
(venv) abeer@Abeers-Air home % python3 main_v3.py
Extracting ....

Feature Extraction: 100%|| 50/50 [14:28<00:00, 17.36s/it]
/Users/abeer/home/venv/lib/python3.13/site-packages/tsfresh/utilities/dataframe_functions.py:198: RuntimeWarning: The columns [' a_x
__mean_abs_change' ' a_x__mean_change'
 ' a_x__mean_second_derivative_central' ...
 ' w_z__permutation_entropy__dimension_7__tau_1'
 ' w_z__query_similarity_count__query_None__threshold_0.0'
 ' w_z__mean_n_absolute_max__number_of_maxima_7'] did not have any finite values. Filling with zeros.
  warnings.warn(
Feature extraction time: 1141.17 seconds
```

Figure 4: Sample Output - Parallel (10 Workers)

workers, it extracts statistical features from time series data stored in a CSV file (rsimu.csv). After cleaning and assigning unique IDs to the data, it runs feature extraction in parallel, imputes missing values, and records the time taken. Then, it standardizes the features, applies KMeans clustering to group the data into three clusters, and visualizes the results using PCA to reduce dimensions to 2D for plotting. This approach is optimized for better speed and efficiency on large datasets. Figure 4 shows the output for the sequential run. For more details about the Parallel run, we have used the code in **appendix B**.

6 Result

6.1 Result Comparison

The sequential and parallel versions of the script both perform time series feature extraction, clustering, and visualization, but they differ significantly in performance. The sequential version processes features using a single CPU core, making it slower and less efficient for large datasets; it is more suitable for small-scale tasks or debugging. In contrast, the parallel version uses tsfresh's MultiprocessingDistributor with 10 workers, enabling concurrent processing across multiple CPU cores. This leads to a significant improvement in speed—typically 2x to 8x faster depending on hardware and dataset size. The parallel version is more appropriate for large-scale data analysis. Overall, the parallel approach is optimized for performance while maintaining the same clustering and visualization pipeline.

6.2 Sequential VS. Parallel

Table 2: Results for Sequential and Parallel

RunType	*Time*
Sequential	1769.88 S
Parallel	1141.17 S

6.3 Clustering

Figure 5 is the output of the plotclusters() function from the code in **appendix A and B**, which visualizes the result of clustering time series features after dimensionality reduction using Principal Component Analysis (PCA).

- Axes (PC1 and PC2): These represent the first two principal components, which are combinations of the original high-dimensional features extracted by tsfresh. PCA reduces these features to two dimensions to allow plotting and visual comparison.
- Dots (Points): Each point corresponds to a sample (or time series segment).
- Colors: The points are color-coded by cluster assignment from the KMeans algorithm. According to the color bar, there are 3 clusters labeled as 0.0, 1.0, and 2.0.

The clustering result shows distinct vertical groupings, which suggests that the time series features are being grouped consistently by pattern similarity. Each vertical strip represents time series data that share similar dynamic properties (like frequency, variance, etc.), which were extracted by tsfresh. This plot effectively demonstrates that the time series data could be grouped into meaningful clusters, and the clusters are visually separable in 2D space, even after dimensionality reduction.

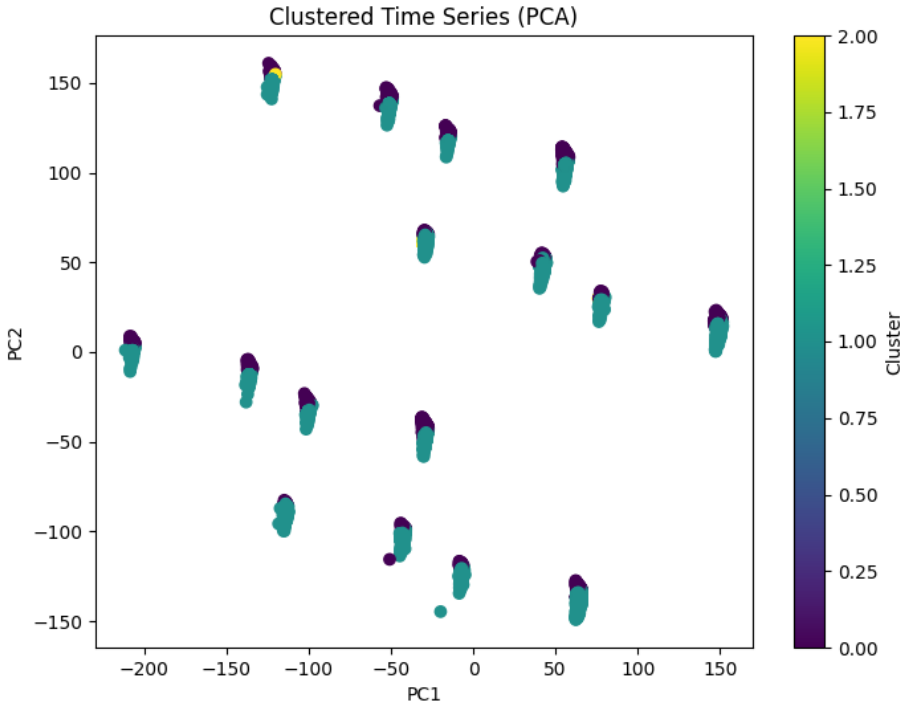

Figure 5: Cluster result

7 Discussion

The comparison between the sequential and parallel implementations highlights the impact of parallel processing on the efficiency of time series feature extraction. While both approaches yield the same analytical results—namely clustering and visualization of time series data—the parallel version demonstrates a clear advantage in performance by leveraging multiple CPU cores through the 'MultiprocessingDistributor'. This optimization significantly reduces execution time, making it well-suited for large datasets or time-sensitive applications. The sequential version, though functionally correct, is limited by its single-threaded execution and is better reserved for small datasets or testing purposes. These findings underscore the importance of parallelization when handling computationally intensive tasks in data analysis workflows. The runtime improved by approximately 35.5%

$$\text{Improvement } (\%) = \left(\frac{1769.88 - 1141.17}{1769.88} \right) \times 100 \approx 35.5\%$$

8 Conclusion

Expanding further, the observed performance difference emphasizes the practical value of optimizing data processing pipelines, particularly in time series analysis where feature extraction can be computationally expensive. In scenarios involving high-frequency or multivariate sensor data—as is often the case in domains like healthcare, IoT, or industrial monitoring—the parallel approach offers scalability and responsiveness that the sequential method cannot match. The use of 10 workers in the parallel implementation significantly reduced the processing time, showcasing how multiprocessing can be an effective strategy to handle larger volumes of data without sacrificing accuracy. Moreover, the consistency in clustering results between both methods confirms that parallelization enhances speed without compromising the quality or integrity of the output. The inclusion of proper time logging in the parallel version also provides better transparency and aids in profiling system performance for further optimization. Overall, this comparison not only validates the benefits of parallel computing in feature extraction but also supports its integration as a standard practice in performance-critical data science pipelines.

9 Appendix A - Sequential

```
import pandas as pd
import numpy as np
import time
from tsfresh import extract_features
from tsfresh.utilities.dataframe_functions import impute
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

def extract_tsfresh_features(df):
    stime = time.time()

    # extract features function take dataFrame, column id,
    #  and group by column sort-time
    extracted_features = extract_features(df, column_id="id", column_sort="t")
    impute(extracted_features)

    etime = time.time()

    time_diff = etime - stime
    print(f"Feature extraction time: {etime - stime:.2f} seconds")

    with open("time_diff.txt","a") as fd:
        fd.write(f"Time Sequential = {time_diff}\n")

    # return the extracted features
    return extracted_features

def cluster_features(features, n_clusters=3):
    # Normalize features
    scaler = StandardScaler()
    features_scaled = scaler.fit_transform(features)

    # Apply KMeans
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    labels = kmeans.fit_predict(features_scaled)

    return labels, kmeans

def plot_clusters(features, labels):

    # Plot the clustered features

    pca = PCA(n_components=2)
    components = pca.fit_transform(features)

    plt.figure(figsize=(8, 6))
```

```
    plt.scatter(components[:, 0], components[:, 1], c=labels, cmap="viridis")
    plt.title("Clustered Time Series (PCA)")
    plt.xlabel("PC1")
    plt.ylabel("PC2")
    plt.colorbar(label="Cluster")
    plt.show()


def main():
    #read data from csv file into DataFrame of class Pandas
    df = pd.read_csv("dataset/rs_imu.csv")

    # Remove null values
    df['id'] = np.arange(0, len(df))

    print("Extracting ....\n")
    ef = extract_tsfresh_features(df)

    print("Clustering ....\n")
    labels , kmeans = cluster_features(ef, 3)

    print("Plotting ....\n")
    plot_clusters(ef,labels)


if __name__ == "__main__":
    main()
```

10 Appendix B - Parallel

```
import pandas as pd
import numpy as np
import time
from tsfresh import extract_features
from tsfresh.utilities.dataframe_functions import impute
from tsfresh.utilities.distribution import MultiprocessingDistributor
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

def extract_tsfresh_features(df,Distributor):

    stime = time.time()

    # Function extract_features takes distributors as a the multiprocesses
    extracted_features = extract_features(timeseries_container=df,
                     column_id='id',
                     column_sort='t',
                     distributor=Distributor)

    impute(extracted_features)

    etime = time.time()

    time_diff =   etime - stime
    print(f"Feature extraction time: {etime - stime:.2f} seconds")

    with open("time_diff.txt","a") as fd:
        fd.write(f"Time Parallel , workers = {time_diff} , 10\n")

    return extracted_features

def cluster_features(features, n_clusters=3):
    # Normalize features
    scaler = StandardScaler()
    features_scaled = scaler.fit_transform(features)

    # Apply KMeans
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    labels = kmeans.fit_predict(features_scaled)

    return labels, kmeans


def plot_clusters(features, labels):

    #Plot the clustered features
```

```
    pca = PCA(n_components=2)
    components = pca.fit_transform(features)

    plt.figure(figsize=(8, 6))
    plt.scatter(components[:, 0], components[:, 1], c=labels, cmap="viridis")
    plt.title("Clustered Time Series (PCA)")
    plt.xlabel("PC1")
    plt.ylabel("PC2")
    plt.colorbar(label="Cluster")
    plt.show()



def main():

    #read data from csv file into DataFrame of class Pandas
    df = pd.read_csv("dataset/rs_imu.csv")

    # remove null values
    df['id'] = np.arange(0, len(df))


    # Create multi-processes for parallelism with no. of 10 workers
    Distributor = MultiprocessingDistributor(n_workers=10,
                                             disable_progressbar=False,
                                             progressbar_title="Feature Extraction")


    print("Extracting ....\n")
    ef = extract_tsfresh_features(df,Distributor)

    print("Clustering ....\n")
    labels , kmeans = cluster_features(ef, 3)

    print("Plotting ....\n")
    plot_clusters(ef,labels)


if __name__ == "__main__":
    main()
```

11 References

[1] Deep learning-assisted Unmanned Aerial Veichle Flight Data Anomaly Detection: A review

[2] Redesign an efficient feature extraction network based on MobileVit, which can simultancly extract global and local information in complex background and learn better feature representation with less computational lose. Also a feature fusion named DSCFF, that can fully interact with the semantic information.Enhancing the detection ability for small object. Finally, the lightweight design of the decoupled head to reduce the computation.

[3] elsevier, A swarm anomaly detection model for IoT UAVs based on a multimodal denoising autoencoder and federated learning

[4] Vibration data-driven anomaly detection in UAVs: A deep learning approach elsevier

[5] arXiv 10.48550/arXiv.2210.09114 http://arxiv.org/abs/2210.09114 arXiv:2210.09114 [cs] Computer Science - Robotics

[5] tsfresh https://tsfresh.readthedocs.io/en/latest/ Feature extraction Time series