Application of Eigenvalues in Movement Identification Using Accelerometer Data

Raynard Fausta - 13524052
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
raynardfaustasmlone@gmail.com - 13524052@std.stei.itb.ac.id

***Abstract*—This article discusses the application of eigenvalues in identifying movement based on accelerometer data. An triaxial accelerometer can measure acceleration in 3 perpendicular planes, X, Y, and Z. Those measurements might seem similar based on its magnitude and nothing can be interpreted from it. However, with the use of eigenvalues of the covariance matrix as compact descriptors, this article successfully builds a supervised machine learning for movement identification. The accelerometer data is first segmented using sliding window technique, and each window is labeled by majority voting. For every window, a 3×3 covariance matrix is computed from mean-centered acceleration samples, and eigenvalue features are extracted to summarize how the variance is distributed across principal directions. The extracted features are then standardized and reduced using Principal Component Analysis (PCA), before being classified using a Random Forest model. With window size = 128, step = 64, PCA components = 8, test size = 0.2, the model successfully achieved an accuracy of 1.0000 based on test split, while 5-fold cross-validation resulted in a mean accuracy of 0.9714 ± 0.0381. When applied to an unlabeled test sequence consisting of 2100 samples, the method produced 31 window predictions. These results indicate that covariance-eigenvalue descriptors, combined with PCA and Random Forest, are effective in identifying basic movements at the window level.**

***Keywords*—acceleration, covariance matrix, eigenvalues, human activity recognition, PCA, Random Forest, sliding window, supervised learning**

I. Introduction

Many devices are constantly recording accelerometer signals from human movement, tracking human movement in daily occasions. It seems that these data only represent acceleration along 3 axes. However, each motion actually produces its own distinct characteristics. Even though one activity might look similar to another, there must be differences in their acceleration patterns.

As every motion has its own characteristics, a useful viewpoint is to treat these data as points in the 3D space. These data do not just have magnitude, but also shape (how spread-out the acceleration is). The data of acceleration of 3 axes can be summarized using the covariance matrix, which contains information about the variance of each axis and the correlation between axes. The eigenvalues of each covariance matrix might be useful in determining basic movements based on how the variance is distributed.

Therefore, the author is interested in analyzing the application of eigenvalues in movement identification using accelerometer data. By focusing on the eigenvalues, key patterns of acceleration can be identified while remaining compact and interpretable. These features can be integrated with machine learning models to perform movement identification, making it suitable for real-time applications.

II. Theoretical Background

A. Acceleration

Acceleration is a vector quantity which has magnitude and direction. Acceleration can only occur when there is change of velocity with respect to time, which is also a vector quantity.

Acceleration is the rate at which the velocity is changing between an interval of time. Therefore, it can be formulated as:

$$a = \frac{dv}{dt}$$

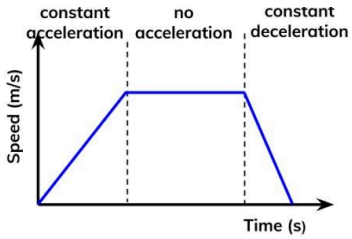

Figure 1. Acceleration [Source: https://senecalearning.com/en-GB/revision-notes/a-level/physics/aqa/4-1-8-graphs-of-motion]

B. Accelerometer

Accelerometer is a sensor for measuring the acceleration of an object. It utilizes an electromechanical sensor to track either static or dynamic acceleration. One of the many types of accelerometers is the triaxial accelerometer. Triaxial accelerometer can measure object's acceleration in three perpendicular planes - X, Y, and Z.

Figure 2. Triaxial Accelerometer [Source: https://environmental.senseca.com/product/hdp356a02-iepe-triaxial-accelerometer-10-mv-g/]

C. Variances

Variances is a statistical measurement which represents how spread-out the data is within a dataset. Variances is represented with the symbol σ^2 and formulated as:

$$\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}$$

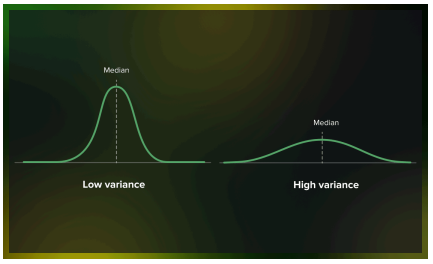

Figure 3. Illustration of Variance [Source: https://serokell.io/blog/bias-variance-tradeoff]

D. Eigenvalues

If A is a matrix with a dimension of n × n, then nonzero vectors defined in R^n are the eigenvectors.

$$Ax = \lambda x$$

The scalar λ is the eigenvalue of A and x is the corresponding eigenvector with λ. Eigenvalue defined the characteristic of matrix with a dimension of n × n.

Eigenvector x is a column vector such that, when multiplied with a matrix with a dimension of n × n, it will result in a vector which is the scalar multiple of x itself.

Therefore, λ is the scalar factor of dilatation. If $|\lambda| > 1$, the vector is stretched; if $0<|\lambda|<1$, the vector is shrunk; if $\lambda<0$, the vector is scaled with reversed direction,

If A is a matrix with a dimension of n × n, the eigenvalues can be calculated with the following steps:

$$Ax = \lambda x$$

(multiply both side with identity matrix I)

$$IAx = \lambda Ix$$

(IAx will remain Ax, identity rule)

$$Ax = \lambda Ix$$

(subtract both side with Ax)

$$Ax - Ax = \lambda Ix - Ax$$

$$\lambda Ix - Ax = 0$$

(distribution rule)

$$(\lambda I - A)x = 0$$

x=0 is the trivial solution of $(\lambda I - A)x = 0$. Hence, the determinant of $(\lambda I - A)$ must be equal to 0 so there are nonzero solutions for $(\lambda I - A)x$. The equation det $(\lambda I - A)x = 0$ is the characteristic equation and the roots of the characteristic equation are the eigenvalues.

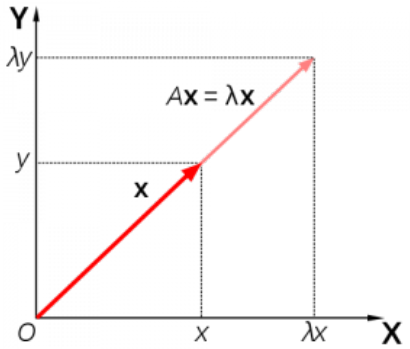

Figure 4. Illustration of Eigenvalue and Eigenvector [Source: https://www.albert.io/blog/what-to-know-about-eigenvalues-and-eigenvectors/]

E. Principal Component Analysis(PCA)

Principal Component Analysis(PCA) is a method that reduces large datasets into principal components which still hold the most original information. The concept of PCA is it transforms the potentially correlated variables into smaller sets of variables (principal components).

PCA is commonly used in machine learning as its ability to obtain the informative features of a large dataset which reduces the model complexity. PCA also eliminates multicollinearity and overfitting in the dataset.

The principal components of PCA are linear combinations of the original variables which have the largest variance compared to other linear combinations. The principal components then served as a transformator for the original dataset into a new coordinate system. Eigenvalues and eigenvectors are used in defining the transformation. Eigenvectors define the direction of the principal components, while eigenvalues determine the variance of each direction.

The first principal component is the direction in which the data has the highest variance. It is best to represent the shape of the projected points. The greater the variability, the more information retained from the original dataset.

The second principal component accounts for the next highest variance. It must be orthogonal and perpendicular to the first principal component or the correlation between both principal components must be zero.

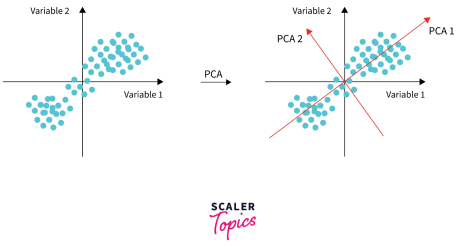

Figure 5. Illustration of Principal Component [Source: https://www.scaler.com/topics/images/concept-of-pca.webp]

F. Covariance Matrix

Covariance matrix is a matrix which is used to represent variance of data in a dataset and covariance values between pairs of elements given in a random vector. It can also be referred to as variance covariance

matrix because the main diagonal of the covariance matrix consists of variance of each element. A covariance matrix is always a square matrix, positive semi-definite, and symmetric. Covariance matrix is a very useful tool for PCA. While the main diagonal element consists of variance, the off-diagonal element consists of covariance. The covariance between two variables can be positive, negative, or zero. Positive covariance indicates positive relationship between both variables, zero indicates no relationship between both variables or they do not vary together, and negative indicates negative relationship between both variables.

The formula of covariance matrix is:

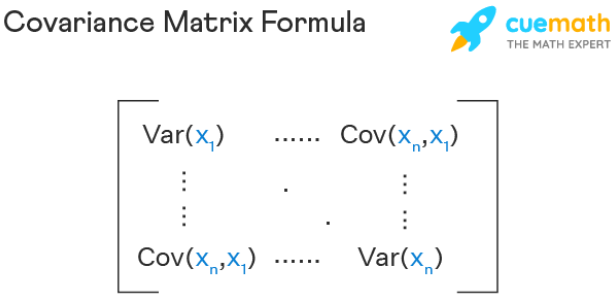

Figure 6. Covariance Matrix Formula [Source: https://www.cuemath.com/algebra/covariance-matrix/]

The formulas for each matrix element are:

Population:

$$Var(X) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n}$$

$$Cov(X) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{n}$$

Sample:

$$Var(X) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}$$

$$Cov(X) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{n-1}$$

G. Centering

Centering is an important aspect for PCA. Centering is done by subtracting each element with the mean value. Centering ensures that the component results are only focusing on the variance within the dataset, and not capturing the overall mean of the dataset as an important variable. Without centering, the first principal component might not be corresponding to the direction which has the highest variance, instead it will correspond to the mean of the dataset.

G. Sliding Window

Sliding window is a problem solving technique which reduces time complexity to O(n) by transforming two nested-loops into a single loop. The main idea of sliding window techniques is the longest sequence or shortest sequence of something that fulfills a given condition.

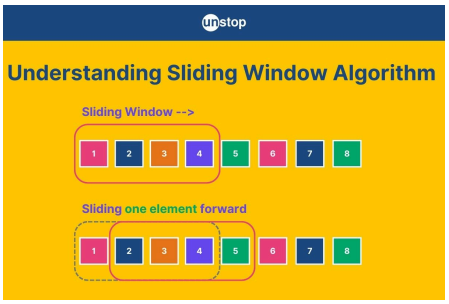

Figure 7. Illustration of Sliding Window Technique [Source: https://unstop.com/blog/sliding-window-algorithm]

H. Random Forest and Decision Trees

Random forest is a machine learning algorithm which combines several results from decision trees to form a final result. It is very commonly used for classification.

The foundation of random forest algorithm is decision tree algorithm. Decision trees consist of nodes used for data separation. Decision trees try to obtain the best separation for data subset. If the observed element fulfills certain criteria, it will follow the "yes" path and if it does not it will follow the alternative path.

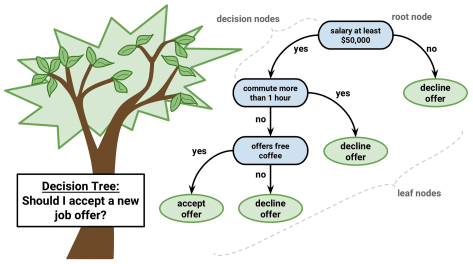

Figure 8. Illustration of Decision Tree [Source: https://blog.gopenai.com/decision-tree-algorithm-484ec33387f9]

Random forest is the extent of the bagging method because it still implements the bagging method and randomized features to create a decision structure that does not correlate. Random forest focuses only on subsets of the feature.

Random forest has 3 main hyperparameters which are size of node, amount of trees, and amount of features retrieved from the sample.

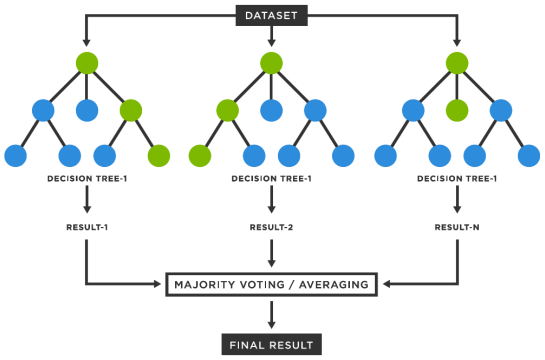

Figure 9. Illustration of Random Forest [Source: https://medium.com/@denizgunay/random-forest-af5bde5d7e1e]

I. Supervised Learning

Supervised learning is a machine learning technique in which the training data are provided with labels (inputs-outputs) in the first place. Model then learns the mapping from input features to the corresponding labels. In the implementation, model utilizes the learnt mapping to classify the input it receives.

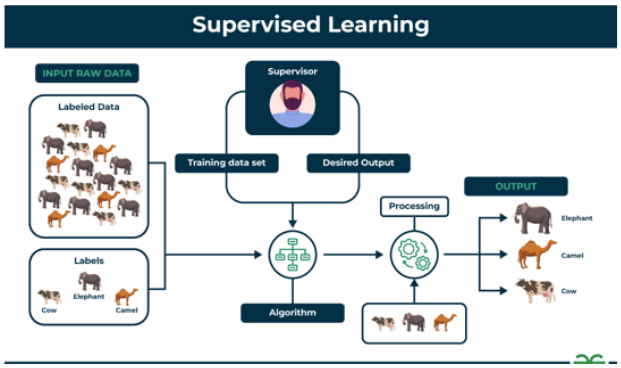

Figure 10. Illustration of Supervised Learning [Source: https://sis.binus.ac.id/2024/10/24/supervised-vs-unsupervised-learning/]

III. Method

A machine learning program in Python will be implemented to identify the type of movement based on accelerometer data. The type of machine learning used in this experiment is supervised learning. As the precondition, there will be a csv file which serves as a labeled training data for machine learning. In the csv file, there is data for 3 basic movements which are still, walk, and shake.

For the movement identification, the program will receive a csv input consisting of acceleration data. The program will then classify those data whether it is still, walk, or shake after running the sliding window technique, centering, building covariance matrix, calculating eigenvalues, PCA, and prediction with Random Forest.

3.1 Training Data Label

Training data provided for machine learning is equipped with input-output pairs. Training data consists of 3 important aspects which are acceleration along the x axis, y axis, and z axis as the input. The program will use this label to learn mapping the acceleration data to the corresponding movement (still, walk, shake).

2.00E-05	0.00597	0.99452	still
-0.01781	-0.00909	0.98017	still
0.0012	0.0268	0.99016	still

Figure 11. Training Data for Still

0.05261	0.11645	0.98746	walk
0.07628	0.10754	1.08086	walk
0.06598	0.1577	1.10892	walk

Figure 12. Training Data for Walk

-0.92781	0.27213	1.37944	shake
0.4466	0.04191	1.39359	shake
-0.02136	-0.05066	0.45259	shake

Figure 13. Training Data for Shake

3.2 Sliding Window

After training data is loaded, the accelerometer data is segmented into overlapping windows. Each segment contains "window_size" consecutive elements each and the next starting index will be shifted forward by "step" elements. The process will continue until there are insufficient elements to make a window. As a result, each window will represent a short time interval of motion which will be used for feature extraction and classification.

```python
def sliding_windows(X: np.ndarray, window_size: int, step: int) -> np.ndarray:
    if X.ndim != 2 or X.shape[1] != 3:
        raise ValueError("Incomplete Data")

    N = X.shape[0]
    if N < window_size:
        return np.empty((0, window_size, 3), dtype=float)

    starts = np.arange(0, N - window_size + 1, step)
    return np.stack([X[s:s + window_size] for s in starts], axis=0)
```

Figure 14. Sliding Window

3.3 Window Labeling

Since one window contains multiple labeled samples, each window will be assigned a label based on which label appeared the most in the window (majority voting). This step ensures that each window has only one ground-truth class to be used for supervised training.

```python
def window_labels(y: np.ndarray, window_size: int, step: int) -> np.ndarray:
    N = len(y)
    if N < window_size:
        return np.array([], dtype=y.dtype)

    starts = np.arange(0, N - window_size + 1, step)
    out = []
    for s in starts:
        chunk = y[s:s + window_size]
        vals, cnt = np.unique(chunk, return_counts=True)
        out.append(vals[np.argmax(cnt)])
    return np.array(out)
```

Figure 15. Window Labeling

3.4 Feature Extraction

After the accelerometer data has been segmented and labeled, feature extraction is done to convert every window into numerical representation. This process will be done based on the eigenvalues of the covariance matrix computed from acceleration data. Firstly, acceleration data in a window will be centered by subtracting the value with the mean value to ensure the computation performed only focuses on variations and not the overall mean. A covariance matrix with a dimension of 3×3 is then calculated to summarize the variance of each axis and their correlation. Eigenvalues $(\lambda_1, \lambda_2, \lambda_3)$ of the covariance matrix are then calculated and sorted in descending order.

To enrich the numerical representation, additional features are also computed from the eigenvalues to strengthen the description about how spread out is the data so that identifying movement times might be more accurate, especially in the case which movement appears to be similar in raw signal magnitude but is different in terms of variability.

```python
def eigen_features_from_window(w: np.ndarray, eps: float = 1e-12) -> np.ndarray:
    wc = w - w.mean(axis=0, keepdims=True)
    C = np.cov(wc.T, bias=False)

    eigvals = np.linalg.eigvalsh(C)
    eigvals = np.sort(eigvals)[::-1]
    l1, l2, l3 = eigvals

    trace = l1 + l2 + l3 + eps
    p1, p2, p3 = l1 / trace, l2 / trace, l3 / trace

    anisotropy = (l1 - l3) / trace
    planar = (l2 - l3) / trace
    linearity = (l1 - l2) / (l1 + eps)

    p = np.array([p1, p2, p3])
    entropy = -(p * np.log(p + eps)).sum()

    r12 = l1 / (l2 + eps)
    r23 = l2 / (l3 + eps)
    r13 = l1 / (l3 + eps)
```

```python
    log_l = np.log(eigvals + eps)

    feats = np.concatenate([
        eigvals,
        np.array([p1, p2, p3]),
        np.array([anisotropy, planar, linearity, entropy]),
        np.array([r12, r23, r13]),
        log_l
    ])
    return feats
```

```python
def extract_features(X: np.ndarray, window_size: int, step: int) -> np.ndarray:
    wins = sliding_windows(X, window_size, step)
    if wins.shape[0] == 0:
        return np.empty((0, 0), dtype=float)

    return np.stack([eigen_features_from_window(w) for w in wins], axis=0)
```

Figure 16. Feature Extraction

3.5 Data Training

After feature extraction, some features might still be correlating with each other. Therefore, PCA is implemented to transform the extracted feature matrix into a more compact representation while still maintaining the most informative variations in the data.

Before applying PCA, the features are standardized so that each feature contributes fairly in the transformation process. PCA then projects the standardized features into a new coordinate system formed by principal components, where the first component has the greatest variance, followed by the second component, and so on. Only the first principal components are maintained and used as the input representation for the classification model.

After the PCA transformation, the training process continues with fitting a Random Forest classifier using the PCA-reduced feature vectors. Each decision tree is constructed from a bootstrap sample of the training windows, and at each split, the tree only considers a random subset of features so that it reduces the correlation and enhances the generalization.

```python
def build_pipeline(pca_components: int, random_state: int = 42):
    return Pipeline([
        ("scaler", StandardScaler()),
        ("pca", PCA(n_components=pca_components, random_state=random_state)),
        ("rf", RandomForestClassifier(
            n_estimators=400,
            random_state=random_state,
            class_weight="balanced_subsample"
        ))
    ])
```

```python
def train_model(X_train: np.ndarray, y_train: np.ndarray, pca_components: int, random_state: int = 42):
    n_features = X_train.shape[1]
    pca_components = max(1, min(int(pca_components), n_features))

    model = build_pipeline(pca_components=pca_components, random_state=random_state)
    model.fit(X_train, y_train)
    return model
```

Figure 17. Data Training

3.6 Data Testing and Evaluation

Testing data for machine learning only contains data of acceleration along x axis, y axis, and z axis. Data undergoes the exact process as in data training. However, the difference is at the final stage where the Random Forest classifier predicts a movement label for each window. As a result, the output of the testing stage is a list of predicted labels at the window level.

The result is then evaluated using accuracy, confusion matrix, and a classification report. Accuracy measures the overall proportion of correctly classified windows. The confusion matrix summarizes how often each true class is predicted, highlighting which movements are often confusing. The classification report provides precision, recall, and F1-score for each class, which describe class-wise correctness and robustness, especially when class distributions are not perfectly balanced.

```python
def predict_file(model, X: np.ndarray, window_size: int, step: int) -> pd.DataFrame:
    XF = extract_features(X, window_size, step)
    if XF.shape[0] == 0:
        raise ValueError("Insufficient sample")

    yhat = model.predict(XF)
    return pd.DataFrame({"window_index": np.arange(len(yhat)), "pred_label": yhat})
```

```python
def test_model(model, X_test: np.ndarray, y_test: np.ndarray):
    pred = model.predict(X_test)

    print("=== Test Evaluation (window-level) ===")
    print(f"Accuracy: {accuracy_score(y_test, pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, pred))
    print("\nClassification Report:")
    print(classification_report(y_test, pred, digits=4))
```

```python
def cv_evaluate(model, X_all: np.ndarray, y_all: np.ndarray, random_state: int = 42, k: int = 5):
    cv = StratifiedKFold(n_splits=k, shuffle=True, random_state=random_state)
    scores = cross_val_score(model, X_all, y_all, cv=cv, scoring="accuracy")
    print(f"=== {k}-Fold CV Accuracy ===")
    print(f"Mean: {scores.mean():.4f}  Std: {scores.std():.4f}")
```

Figure 18. Data Testing and Evaluation

IV. Results and Discussion

4.1 Results:

Window size: 128

Step: 64

PCA Components: 8

Test Size: 0.2

Seed: 42

```
C:\Users\LENOVO\Downloads>python movement_eigen_accel.py --csv train_accel_eigen_ready.csv --predict_csv test_accel_eigen_ready.csv --window_size 128 --step 64 --pca_components 8 --test_size 0.2 --seed 42
Test Evaluation (window-level):
Accuracy: 1.0000

Confusion Matrix:
[[6 0 0]
 [0 6 0]
 [0 0 9]]

Classification Report:
              precision    recall  f1-score   support

       shake     1.0000    1.0000    1.0000         6
       still     1.0000    1.0000    1.0000         6
        walk     1.0000    1.0000    1.0000         9

    accuracy                         1.0000        21
   macro avg     1.0000    1.0000    1.0000        21
weighted avg     1.0000    1.0000    1.0000        21

5-Fold CV Accuracy:
Mean: 0.9714  Std: 0.0381
```

```
Predictions (window-level):
 window_index pred_label
            0       still
            1       still
            2       still
            3       still
            4       still
            5       still
            6       still
            7       still
            8       still
            9        walk
           10        walk
           11        walk
           12        walk
           13        walk
           14        walk
           15        walk
           16        walk
           17        walk
           18        walk
           19        walk
           20        walk
           21        walk
           22       still
           23       shake
           24       shake
           25       shake
           26       shake
           27       shake
           28       shake
           29       shake
           30       shake
Total predicted windows: 31
```

4.2 Discussion

The proposed pipeline was evaluated at window level using a stratified holdout split on window level. After the sliding window is implemented and majority-vote labeling, the resulting window samples are split into training and testing subtests with the size of the testing = 0.2. Based on the dataset, there are 21 windows for the hold out test.

For the holdout test set, the model successfully gained 1.0000 accuracy. Indication from the confusion matrix shows perfect classification across the tree movements provided in data: 6/6 for shake, 6/6 still, and 9/9 for walk.

Consistently with this, classification report shows a precision, recall, and F1-score of 1.0000 for every class. This proves the additional features extracted from the eigenvalue are highly separable for the targeted movements.

In addition, to assess robustness beyond a single split, 5-fold stratified cross-validation is performed on the complete window-level training dataset. The model achieved a mean accuracy of 0.9714 with a standard deviation of 0.0381. Compared to the perfect holdout result, cross-validation indicates that performance can vary slightly depending on which fold is used as the test subset; however, the overall results remain consistently strong.

The absence of the off-diagonal element in the confusion matrix indicates that all windows are classified correctly and there are no wrong predictions between classes. This strengthens the claim that those 3 movements are differentiable by the model. Therefore, eigenvalue-based descriptors can differentiate movements even when raw acceleration magnitudes may look similar.

After performing data training, the same pipeline was applied to the unlabeled test sequence which consists of 2100 data and segmented into 30 windows. The predicted sequence indicates still at windows 0-8, walk at windows 9-21, still at window 22, and shake at windows 23–30.

While the holdout evaluation achieved perfect accuracy, the 5-fold cross-validation does not do so (mean 0.9714, standard deviation 0.038**1**). This indicates that performance can vary slightly across different splits. In addition, this experiment only considers three movement classes (still, walk, and shake) and assigns a single label per window using majority voting. Therefore, windows near transition boundaries may be ambiguous in a more vast case.

V. Conclusion

This article demonstrates that covariance-eigenvalue–based features can provide an effective and interpretable representation for movement identification from triaxial accelerometer data. By combining sliding-window segmentation, mean-centering, covariance computation, eigenvalue feature extraction, PCA-based compaction, and Random Forest classification, the proposed pipeline achieved perfect performance on the holdout window-level test split (accuracy 1.0000) and consistently strong results under 5-fold cross-validation (mean accuracy 0.9714 with standard deviation 0.0381).

When applied to an unlabeled data, the model successfully produced 31 window-level predictions.

However, the biggest concern is that the movements that can be identified are only limited to 3 basic movements (still, walk, and shake) in this experiment. Therefore implementation in a larger dataset or more various movement classification might be ambiguous.

VI. Appendix

Video Link:
https://drive.google.com/file/d/1JhjlozvGxLzFMN9T6Ayh6M_3pTZQxqr8/view?usp=sharing
https://youtu.be/GcsBatlenmI
Duration: 24 minutes 55 seconds

Programs used:
https://github.com/RayapSunggal/Makalah-Algeo.git

VII. Acknowledgment

The author sincerely thanks God Almighty for providing the strength and opportunity to complete this article successfully. The author also extends his deep appreciation and gratitude to Dr. Ir. Rinaldi, M.T. of K01 IF2123 Linear and Geometric Algebra, whose semester-long support enabled the smooth completion of this article. The author would also like to extend his personal appreciation to creator of 芒种(Grain in Ear) song, whose music has accompanied and uplifted the author during the preparation of this article

References

[1] D. Halliday, R. Resnick, and J. Walker, *Fundamentals of Physics*, 10th ed. Hoboken, NJ, USA: Wiley, 2014.

[2] A. Hayes, *What Is Variance in Statistics? Definition, Formula, and Example*, Investopedia, 2025. [Online]. Available: https://www.investopedia.com/terms/v/variance.asp. [Accessed: Dec. 24, 2025].

[3] Rinaldi, *Nilai Eigen dan Vektor Eigen*, STEI, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-19-Nilai-Eigen-dan-Vektor-Eigen-Bagian1-2025.pdf. [Accessed: Dec. 24, 2025].

[4] IBM, *What is Principal Component Analysis (PCA)?*, IBM. [Online]. Available: https://www.ibm.com/think/topics/principal-component-analysis. [Accessed: Dec. 24, 2025].

[5] Cuemath, “Covariance Matrix - Formula, Examples, Definition, Properties,” Cuemath. [Online]. Available: https://www.cuemath.com/algebra/covariance-matrix/. [Accessed: Dec. 24, 2025].
[6] G. Ershad, *Sliding Window Algorithm Explained*, *Built In*, 2025. [Online]. Available: https://builtin.com/data-science/sliding-window-algorithm. [Accessed: Dec. 24, 2025].
[7] Sydney Firmin, *Tidying up with PCA: An Introduction to Principal Components Analysis*, Medium, 2019. [Online]. Available: https://medium.com/data-science/tidying-up-with-pca-an-introduction-to-principal-components-analysis-f876599af383. [Accessed: Dec. 24, 2025].
[8] E. Kavlakoglu, *Apa itu Random Forest?*, IBM. [Online]. Available: https://www.ibm.com/id-id/think/topics/random-forest. [Accessed: Dec. 24, 2025].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri dengan bantuan Generative AI untuk penyempurnaan tata bahasa, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

Raynard Fausta
13524052