ROBOT JOINT MOVEMENT DETECTION USING PRINCIPAL COMPONENT ANALYSIS

CHENAB

Applied and Computational Mathematics, University of Washington, Seattle, WA
chenab@uw.edu

Abstract. The following paper discusses how the problem of designing a real-time algorithm to recognize the movements, namely walking, jumping and runnning, of the OptimuS-VD humanoid robot. The data available was movements of the joints recorded as Euler angles. The main method used was Priciple Component Analyis (PCA) for dimensionality reduction and visualization of the movements in lower dimensions. Then centroids of the dimensionally reduced three types of data data were used in the classification algorithm.

1. Introduction and Overview

The main problem was developing a method of classifying movement data from the robot into three types : walking, running and jumping. The available data was 5 samples of each of the three types of movement. The samples were recorded for 100 time steps, each 1.4 seconds long and the overall data was saved as a 114×100 matrix, in a very high-dimensional space.

The most important aspect of the problem was dimensionality reduction of the highly complex movement data. I applied Principle component analysis to get the orthogonal spatial modes representing the different kinds of dominant spatial patterns in the data. An analysis was conducted to determine how many modes were needed to capture various percentages of the original data's energy (using the Frobenius norm). Further visualizations were made in the low dimensional 2D and 3D PCA space.

Further more, for different k modes, the centroids in the PCA space were calculated for the three types of movement. And each sample in the data set was classified according to the closest distance to the centroid. Then finally, the classification algorithm was tested on new test samples, and the final accuracy achieved was close to 95.33 %. The following sections will go over the details of the key concepts and implementation.

2. Theoretical Background

The **Singular Value Decomposition** essentially decomposes any matrix $A \in R^{n \times m}$ into the following:

$$A = U\Sigma V^T$$

U is a unitary matrix with $U \in R^{n \times n}$

V is a unitary matrix with $V \in R^{m \times m}$

Σ is a diagonal matrix $\Sigma \in R^{n \times m}$ with all elements assumed to be non-negative and ordered from largest to smallest magnitude.

The SVD decomposition of the matrix A essential applies a unitary transformation preserving the unit sphere via V^*, followed by the creation of an ellipse with principal semi-axis given by the matrix Σ. This is done through a stretching operation. Finally, the (hyper) ellipse is rotated by a unitary transformation that is caused by the matrix U.

The **Covariance Matrix**: The **covariance** between data sets essentially tells us how related two quantities are. This indirectly tells us how much **redundancy** is present in our dataset, since covariance measures the statistical dependence between two variables.

Date: March 25, 2025.

The **Covariance Matrix** for a data, say $X = [x_a, y_a, x_b, y_b...]$ is given by :

$$C_x = \frac{1}{n-1} XX^T$$

This matrix has several key properties:

- It is square $m \times m$ and symmetric.
- It's diagonals give the variance of the measurements
- It's off-diagonal elements represent the covariances between different measurements.

Essentially, it captures the correlations between all possible pairs of measurements from our data. Ideally, we want our data to have as little strongly statistically dependent variables (high covariance).

Principal Component Analysis is a dimensionality reduction technique that is used for **removing redundancy** and noise. It involves reducing the number of variables in a data set while capturing **as much information** (represented by variance) about the system as possible. We also get information about the most important directions along which the data varies the most.

PCA involves diagonalizing the covariance matrix using the **SVD**. Essentially, we define a transformed variable $Y = UX$, where U is the unitary transformation seen in the SVD : $X = U\Sigma V^T$. Computing

$$C_Y = \frac{1}{n-1} YY^T$$

eventually simplifies to

$$C_Y = \frac{1}{n-1} \Sigma^2$$

This also shows that $\Sigma^2 = \Lambda$. Where Λ is a diagonal matrix whose entries correspond to the distinct eigenvalues of XX^T, thus giving a relation between the **singular values** and the **eigenvalues**.

We will be using PCA for the following key reasons:

- To reduce redundancy in our robot movement data set, while capturing as much information as possible
- Increasing performance and efficiency by dimensionality reduction
- Visualizing high dimensional robot joint data in 2D and 3D to get intuition.

An important concept we will use to gauge the amount of information being captured is the **Frobenius Norm**. Frobenius norm essentially captures the total variance (or energy of the system). It is given by

$$||A||_F = \sqrt{\sigma_1^2 + \sigma_2^2 + ... + \sigma_r^2}$$

where σ represents the singular values ordered by magnitude

3. Algorithm Implementation and Development

The following algorithm was implemented to build a lower dimensional projection of the robot data, visualize the movement, and build a classification algorithm to recognize movements.

1.) Initial Setup and Data Processing

We first load the 5 training samples for each distinct movement: walking, jumping and running with shape 114×100. First dimension records the location of the joints and the second records the timesteps. Then we compile all the samples (15 in total) into a single matrix X_{train} of shape 114×1500 using the **hstack** function from the **numpy** library.

2.) Obtaining Principal Components

Next, we use the **PCA fit** function from the **sklearn** library to fit the X_{train} dataset. First of all, it centers the X_{train} to be mean zero. This is very important because without centering, the mean value can skew the covariances leading to components that do not reflect the mean rather than the intrinsic variation in the data. After centering, it uses the **SVD** on the centered data matrix to decompose into U (which contains the left singular vectors, that are the eigenvectors of the covariance matrix of X_{train}), Σ, diagonal matrix containing the singular values sorted in descending order, and V^T containing the right singular vectors transposed. The PCA modes thus are spatial modes.

3.) Calculating percentage energy using Frobenius norm and plotting The frobenius norm captures the amount of variance in the data. Therefore, we use to to get the minimum number of modes we need to capture certain percentages of the original information/variance.

This is essentially done by finding the ratio: $\frac{\sigma_1^2+\sigma_2^2+...+\sigma_k^2}{\sigma_1^2+\sigma_2^2+...+\sigma_r^2}$ Where k is the minimum modes needed to get a said percentage, and the denominator represents the total energy by including all the singular values. We use the **PCA singular values** to get the σs and **argmax** function from the **numpy** library to get modes for which we exceed the thresholds

Finally, we plot this energy percentage vs the number of k modes, with values horizontal lines indicating the thresholds for 70%, 80%, 90% and 95% respectively

4.) Visualizing Robot Movement data in 2D and 3D We fit and transform the X_{train} using **PCA fit transform** into the spatial modes for k = 2 and k = 3. Then we assign colors to each distinct movement (walking, running and jumping) and plot the projected movements from higher dimenstions into the lower dimensions of 2 and 3 respectively using **matplotlib**

5.) Creation of the Ground Truth Vector and Centroid Computation

Next we create a ground truth vector that essentially stores an integer per class. So for each sample, we assign it a value of 0(Walking), 1(Jumping) and 2(Running).

We also compute the centroids for any given k in the k-modes PCA space. This is done by **averaging** their coordinates in each dimension in the PCA space respectively.

5.) Main Classification Algorithm

We then create a vector of **trained labels**. Each sample in X_{train} is assigned a label depending on whichever centroid is the **closest** to it in the k-mode PCA space using **linalg norm** from the **numpy** library. This is done for various k values and the accuracy is calculated for each k value using the **accuracy score** function from **sklearn**. A plot is made for each k value vs accuracy score.

6.) Testing our Algorithm on New Test Samples We test our classification algorithm on the test sample by assigning the ground truth labels in similar fashion as the training set and then projecting X_{test} onto the K-PCA space using **PCA fit** and assigning labels based on our previously computed centroids and compute the accuracy scores for various k values.

4. Computational Results

Upon investigation, it was found that the following number of PCA modes were needed to reach the threshold energies in terms of the Frobenius Norm:

```
PCA modes needed to approximate up to 70% of the energy: 2
PCA modes needed to approximate up to 80% of the energy: 3
PCA modes needed to approximate up to 90% of the energy: 5
PCA modes needed to approximate up to 95% of the energy: 7
```

And we get the following graph:

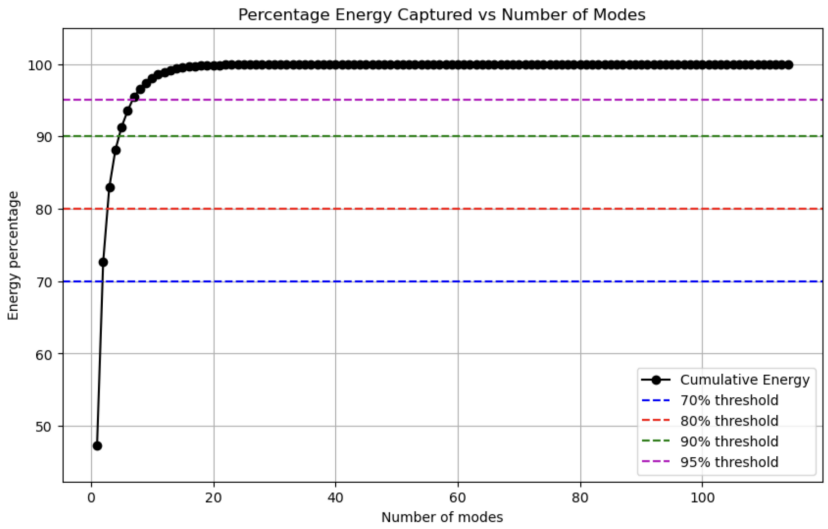
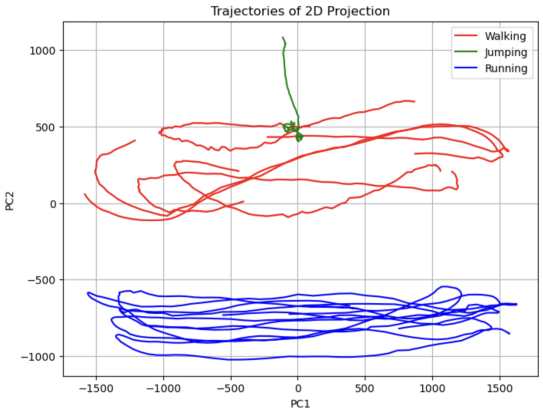


FIGURE 1. Plot of the Energy percentage in terms of the Frobenius norm vs k modes in PCA space

Now we see the visualization of the projection of X_{train} onto 2 and 3 PCA modes respectively.



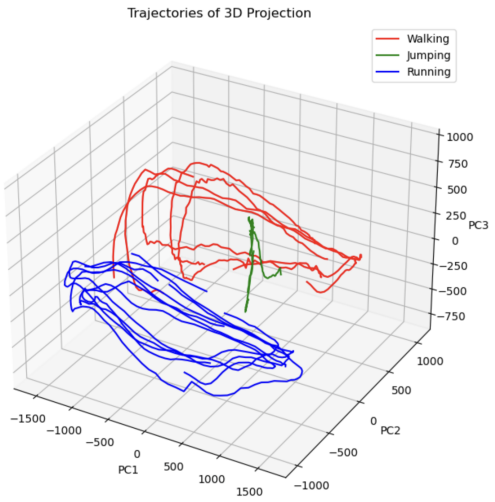


FIGURE 2. Projection of Trajectory in 3 PCA modes

These visualizations demonstrate a clear **distinction** between the three types of movements :walking, jumping and running when projected into the lower dimensional K-PCA space. Jumping, which are unique in that they involve predominantly vertical movement , is clearly distinct from walking and running being more spread out along the Y-axis in the 2D projection and the Z-axis in the 3D projection, Whereas running and walking ,which mostly happening at the ground level with no change in height, show similar distribution pattern in the projected spaces.

Moving on, we compute the centroids for each movement for different values of K modes in the PCA space. A visualization is provided below for the centroids in 2 dimensions:

```
Centroids in 2D PCA space:
Walking centroid (label 0): [-36.88211143 253.35282541]
Jumping centroid (label 1): [-23.88986635 499.36826149]
Running centroid (label 2): [  60.77197779 -752.7210869 ]
```

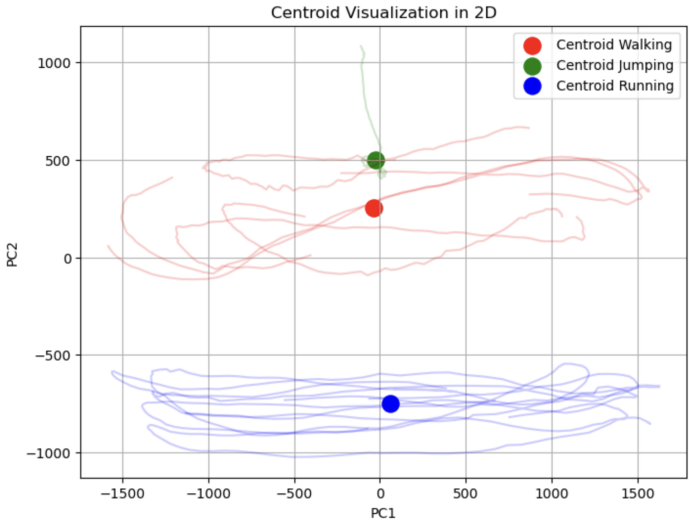


FIGURE 3. Plot showing the centroids in 2-PCA space for the training data

Next, we will use these centroids as the basis for our classification. Assigning each sample a label based on the centroid it is closest to in the K-PCA space. We get the following variation in accuracy of our training data for various values of k: Accuracy(K = 2) = 88.3%; Accuracy(K = 5) = 75.07%; Accuracy(K = 10) = 88.8%; Accuracy(K = 20) = 91.07%; Accuracy(K = 50) = 91.07%

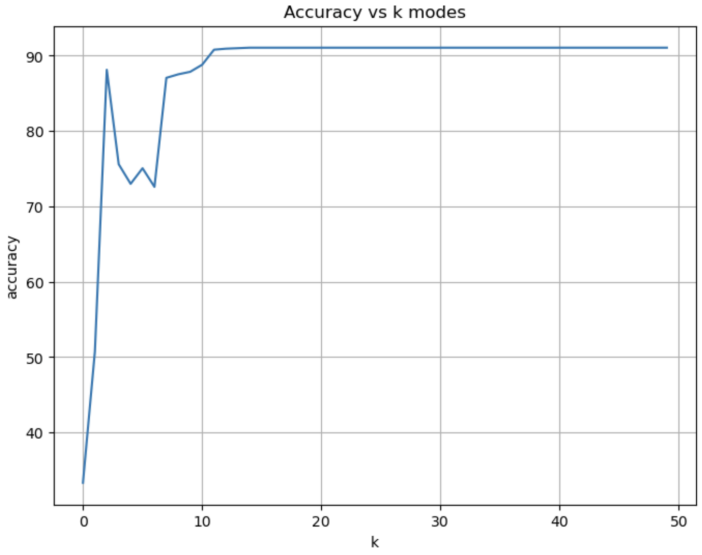


FIGURE 4. Plot showing the variation in accuracies for different number of modes in the PCA space. This is for the training dataset

We notice that the accuracy initially increases then decreases for smaller values of K between 0 and 10. It shows no clear pattern initially and its variation is akin to random pertubations. The accuracy then increases monotonically from k = 6 onwards but eventually converge to a peak accuracy of 91.07%. The initial random behaviour for accuracy vs k can be attributed to the sensitivity of our classifier algorithm

to the initial conditions, which are the positions of our centroids. Their positions are highly vulnerable to outliers, and the distribution of outliers greatly varies for smaller modes, influencing the accuracy. However, as the number of modes increases, the probability of individual outliers greatly influencing the centroid position greatly diminishes and we observe a convergence in accuracy.

As far as optimal K is concerned, I believe K = 2 should be the most optimal. It is 88% accurate compared to the 91.07% peak accuracy. So there is a huge beneficial trade-off in the number of modes, as we remove maximum redundancy from our data. But at the same time we are not compromising on accuracy that much, so a win-win.

Next, we run the same algorithm for our test data, but we use the centroids previously computed from our training data. The following accuracies are observed for different values of K: Accuracy(K = 2) = 98.33%; Accuracy(K = 5) = 91.67%; Accuracy(K = 10) = 94.33%; Accuracy(K = 20) = 95.33%; Accuracy(K = 50) = 95.33% We see the same disturbances initially because of the data being vulnerable to outliers and the accuracy similarly converges but to a different value of 95.33%. A surprising result is that the accuracy for K = 2 is a staggering 98.33%, making it an ideal choice once again.

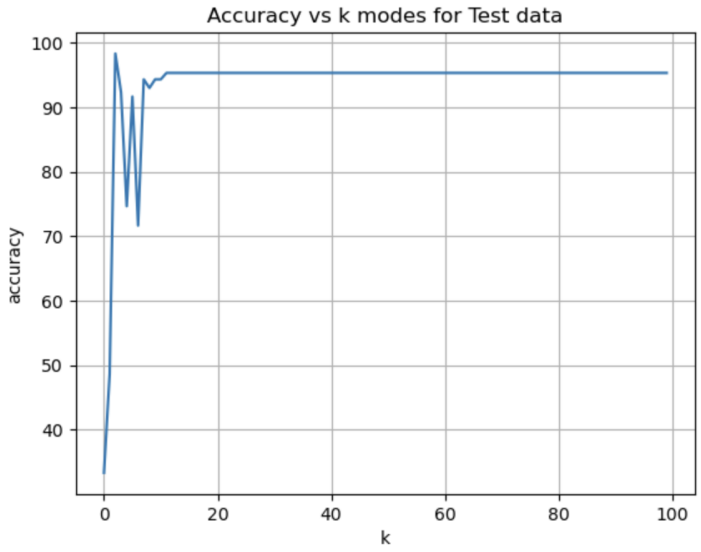


FIGURE 5. Plot showing the variation in accuracies for different number of modes in the PCA space. This is for the Test dataset

5. Summary and Conclusions

In conclusion, we effectively used Principal Component Analysis and Classification based on centroids to solve the problem of projecting high-dimensional robot joint data in lower dimensions, and succesfully implementing a classification algorithm achieving high accuracy for the test data (98.33%). We also plotted the accuracy for various k values as well as analyzed robot trajectory in 2D and 3D/

We also used the KNN(K-nearest neighbors algorithm) for an alternative classifier. In this we choose an arbitrary number of neighbors (5 in my implementation). Then for each sample, we compute the euclidean distance of that point in PCA space with every other point. Then we choose the K points closest to that point (5 in my implementation). And classification is done based on whichever is most common by majority. I got the following results:
For **k = 2**: KNN Train Accuracy: 99.33%, KNN Test Accuracy: 96.67%
For **k = 5**: KNN Train Accuracy: 100%, KNN Test Accuracy: 99%
For **k = 10**: KNN Train Accuracy: 100%, KNN Test Accuracy: 100%
For **k = 20**: KNN Train Accuracy: 100%, KNN Test Accuracy: 100%
For **k = 50**: KNN Train Accuracy: 100%, KNN Test Accuracy: 100%
The results speak for themselves. KNN is way more powerful and accurate than our centroid method. Even

with a small number of modes, KNN achieves 100% accuracy. I believe this is because KNN is very effective when there are highly defined clusters as in our problem.

References

1.) Kutz, J. Nathan. Data-Driven Modeling Scientific Computation: Methods for Integrating Dynamics of Complex Systems and Big Data. Oxford: Oxford University Press, 2013. Chapter 15, "Matrix Decompositions," 375-390. Sections include "The Singular Value Decomposition (SVD)," 375-378; "The SVD in Broader Context," 379-383; "Introduction to Principal Component Analysis (PCA)," 384-386; "Principal Components, Diagonalization and SVD," 387-389; "Principal Components and Proper Orthogonal Modes," 390. Chapter 17, "Unsupervised Machine Learning": "Data Mining, Feature Spaces and Clustering" (p. 443) "Unsupervised Clustering Algorithms" (p. 447)