"A comparative analysis across algorithmic, machine learning, and visual paradigms for the automatic detection of the perceived origin of full-body human movement"

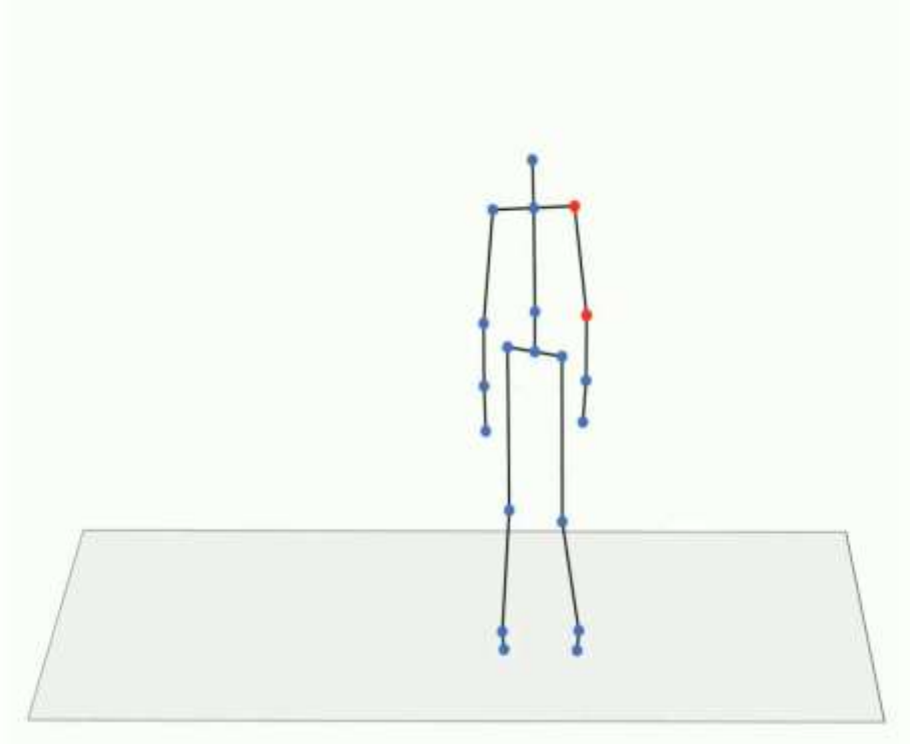
Master Degree Thesis
Computer Engineering

Candidates:
Fausto Martina – 4515876
Romano Gabriele – 5383588

Advisors:
Prof. Volpe Gualtiero
Prof. Oneto Luca

Università di Genova

Origin of movement



Problem statement

What we want to do

Develop new methods for estimating and classifying the origin of movement

Why this is important

Improve sport performances

Injuries prevention and rehabilitation

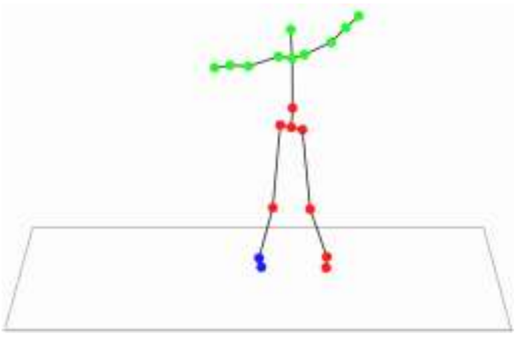
Perform self-assessment and correct movements independently, without the need for a coach's presence

Innovativeness

Visualization approach enables considerations on the origin of movement and also on multiple origins

Machine Learning approach that has not been implemented before

Outline of our work

Input	Feature extraction	Model	Output	Approach
Raw data	Trajectory's derivatives (velocity, etc.)	Graph Clustering	Multiclass classification on the whole body	**Algorithmic**
Dataset		Perfect Matching		**Visualization**
	Statistical descriptors of the trajectories	Random Forest	Binary classification on specific edges	**Machine Learning**

Raw Data

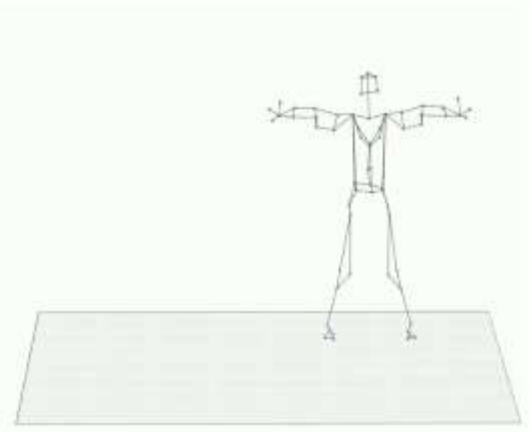
Video Recordings

Fragments selection

We selected short segments where we observed the OoM

Validation

The annotation and the validation of the OoM was made by us, the professor and an experienced dancer



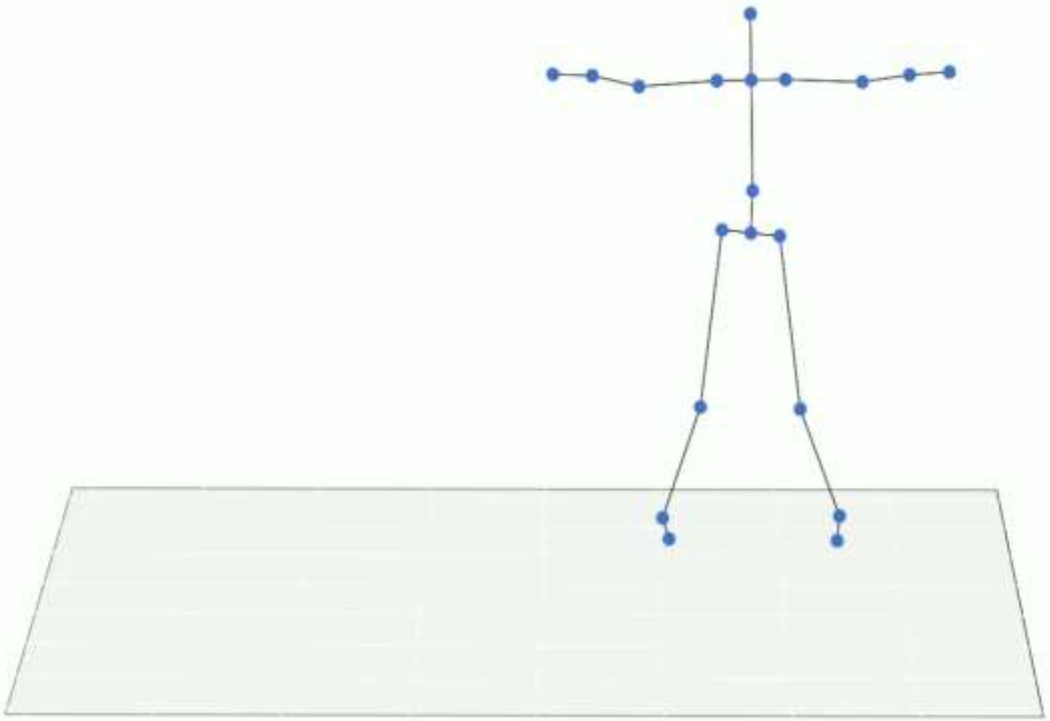
MoCap

Markers Compression

We compressed the markers from 64 and 41 into a skeleton with 20 nodes

Dataset

Skeleton model



Classification

There are 19 possible classes corresponding to the edges of the skeleton and each sample is characterized by a single class

Dataset composition

60 time series of each of the 20 joint in three-dimensional space

Distribution

The dataset in unbalanced and
4 classes are never assumed in the dataset

Algorithmic Approach

What it is

A method that compares features such as velocity, acceleration, and angular momentum of various joints, classifying nodes physically connected with more dissimilar feature values as the origin of the movement

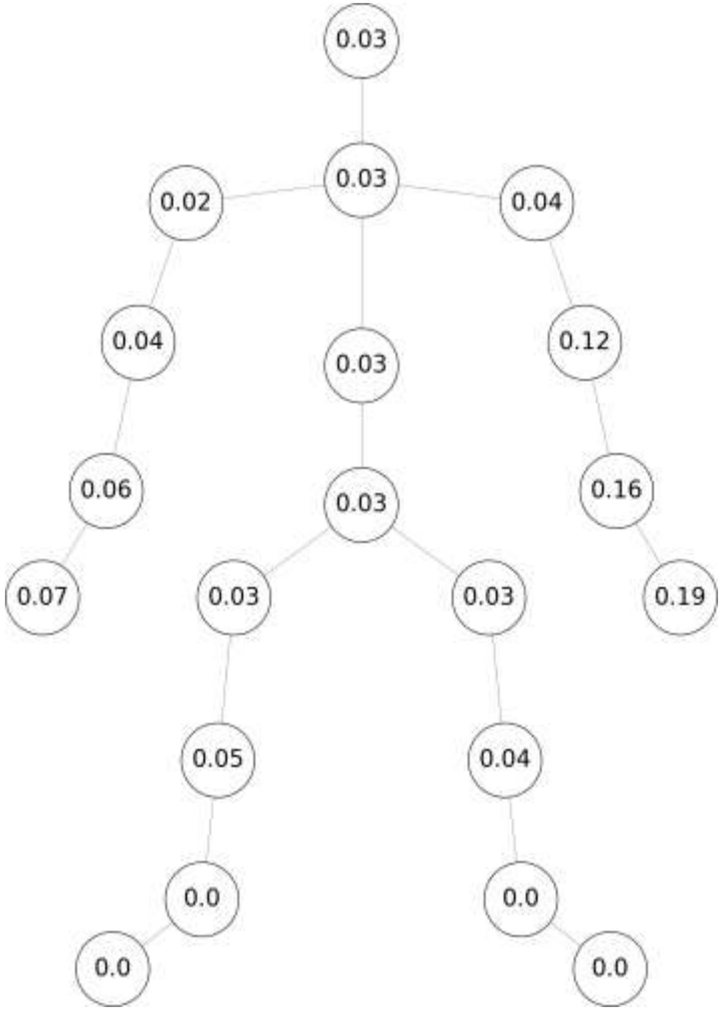
What it is for

It can potentially provide multiple OoM if the candidate joints for OoM are not physically linked

Algorithmic Approach

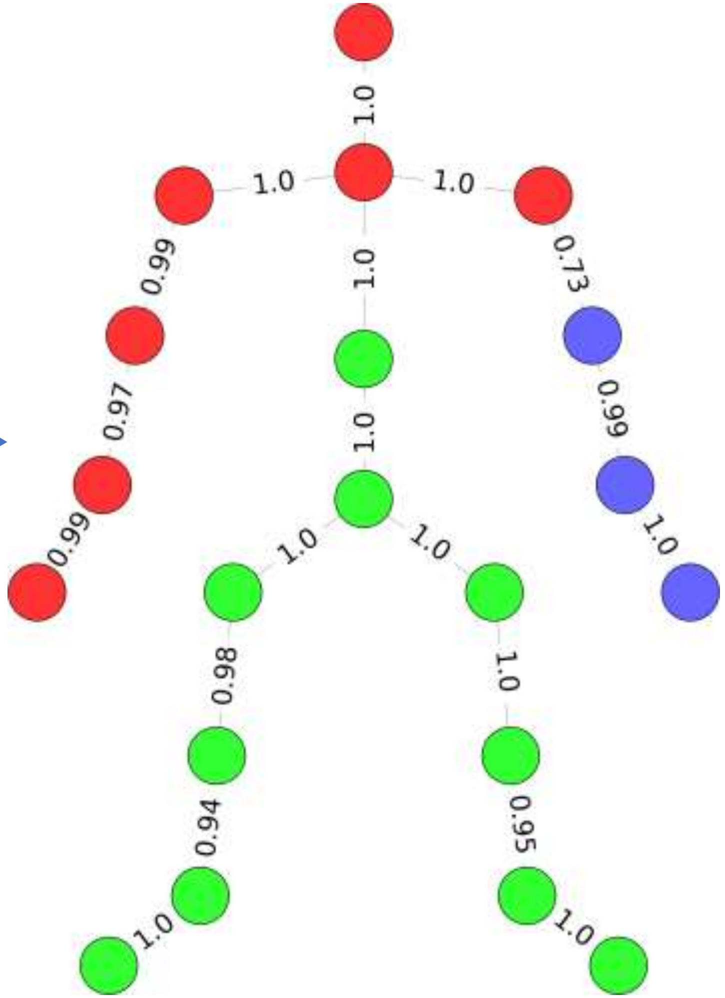
Feature extraction

Features (velocity, acceleration, angular momentum) extraction for every joint



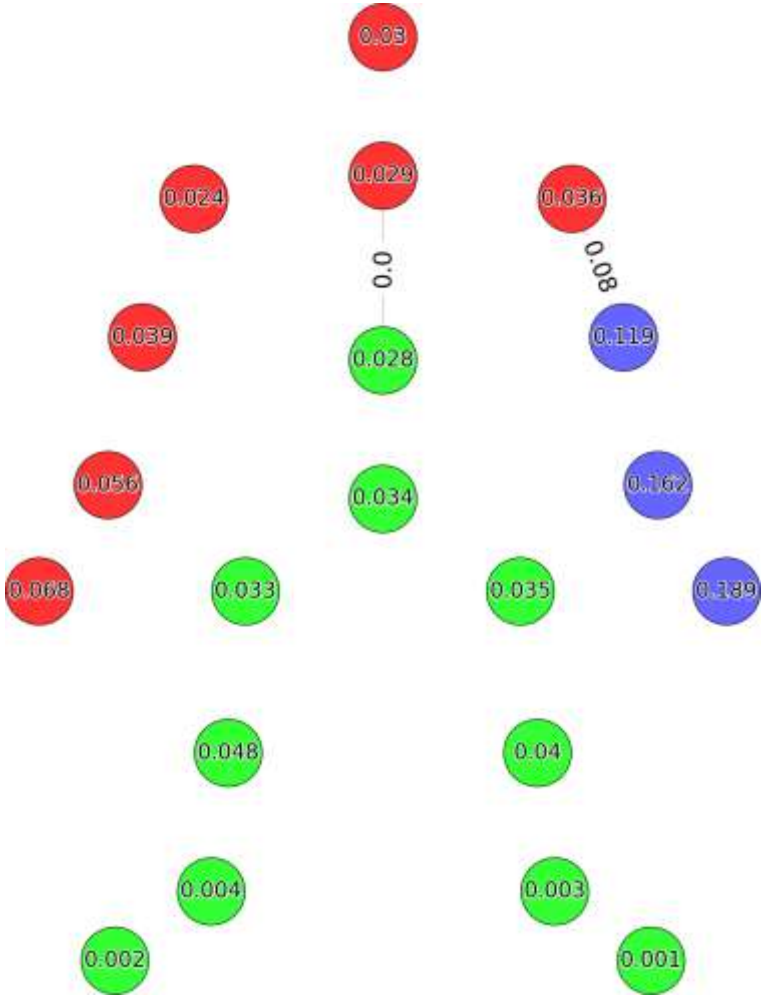
Spectral clustering on the adjacency matrix

- Cosine Similarity
- Shi-Malik spectral clustering



Auxiliary Graph

Euclidean distance is calculated on adjacent nodes of different clusters to find edges' weight



Algorithmic Approach

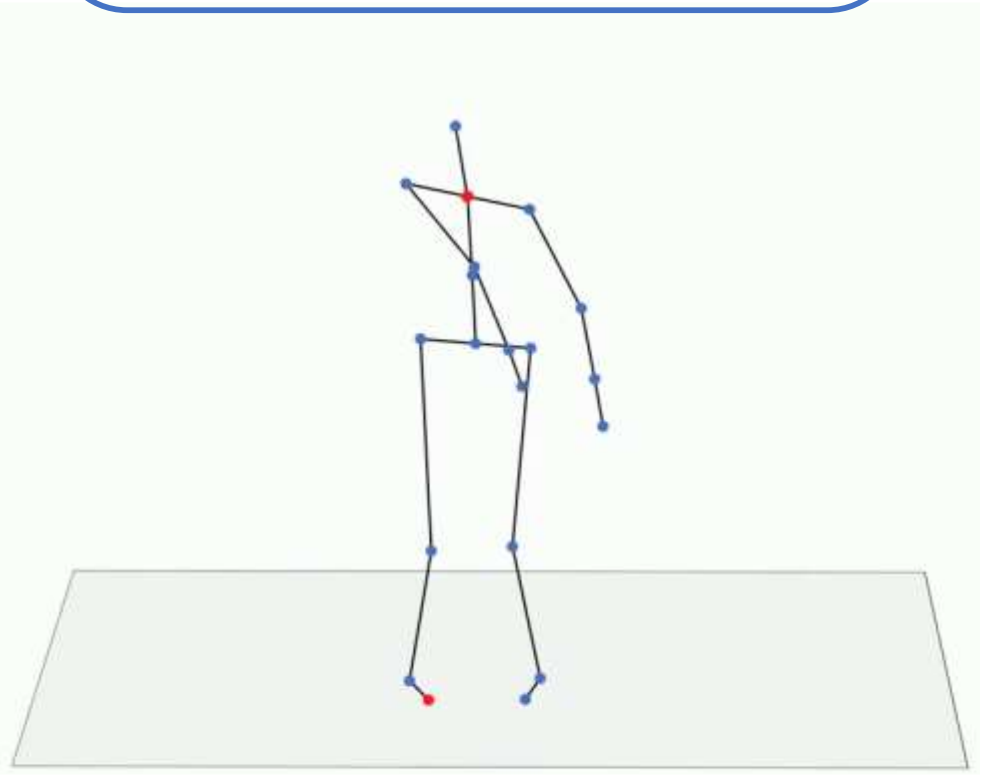
Weighted Degree Centrality calculated for each frame

→

Once all frames are processed, the two most frequent maximums across all frames are selected

→

When compared to the ground truth, it will be considered correct if one of the two nodes is at one of the two ends of the edge identified as OoM



Visualization Approach

What it is

Approach used to make considerations on OoM by visualizing how clusters grouping joints with similar characteristics change during the execution of a movement

What it is for

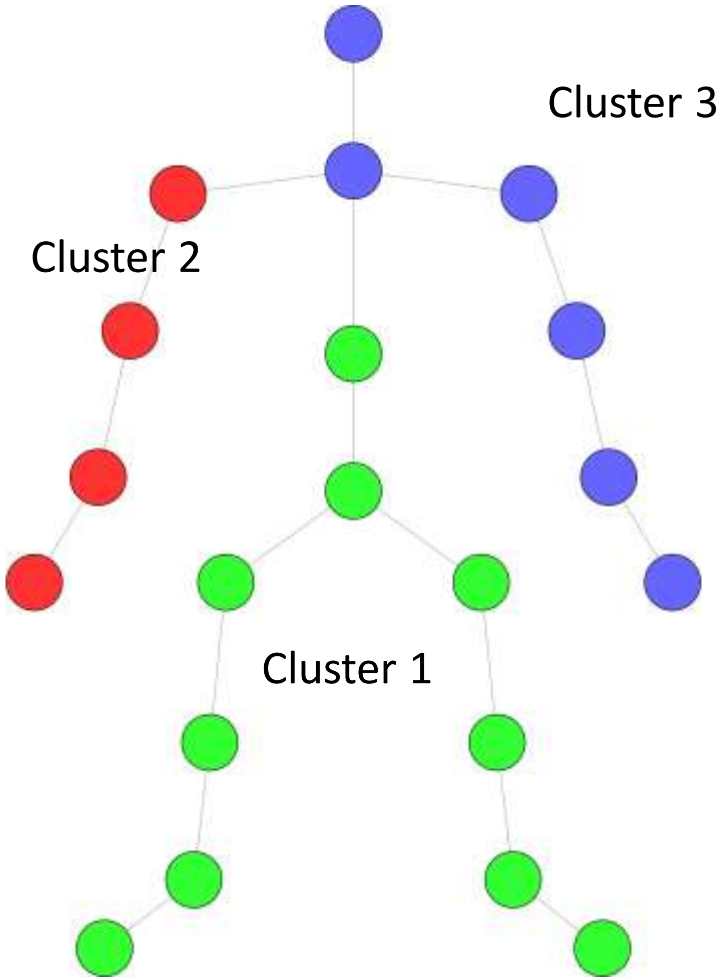
It can be useful to better visualize which parts of the body move for example at the same velocity during a motion and how changes in velocity occur to initiate specific movements

Method

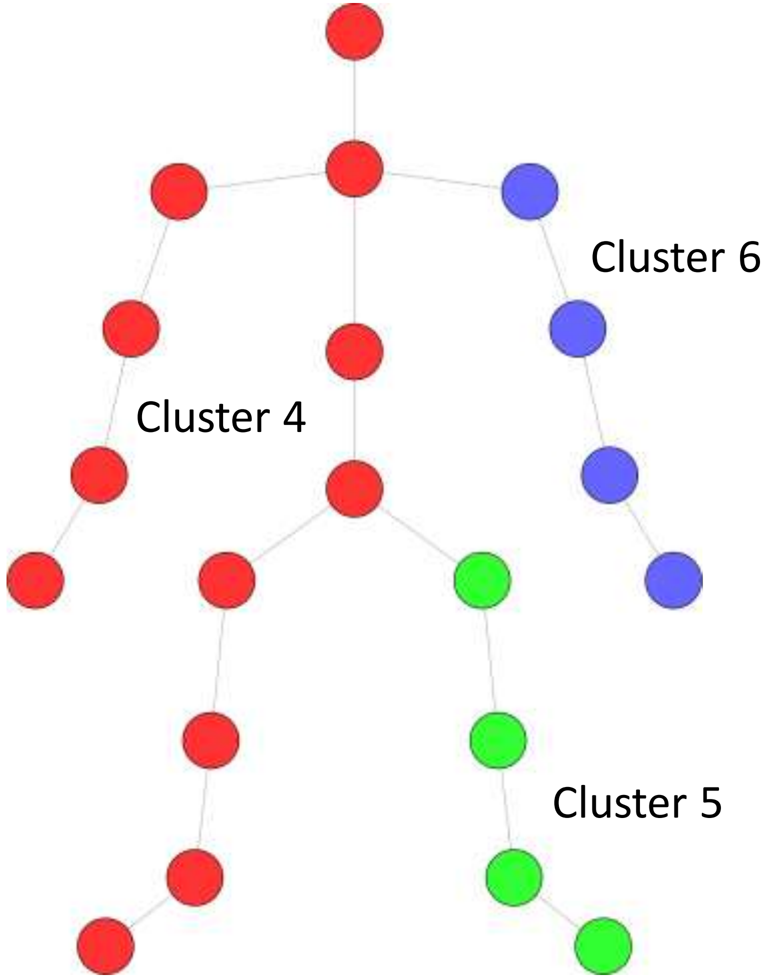
Setting a Maximum Weight Perfect Match problem and solving it applying the Hungarian Algorithm

Visualization Approach

Initial clusters assignement

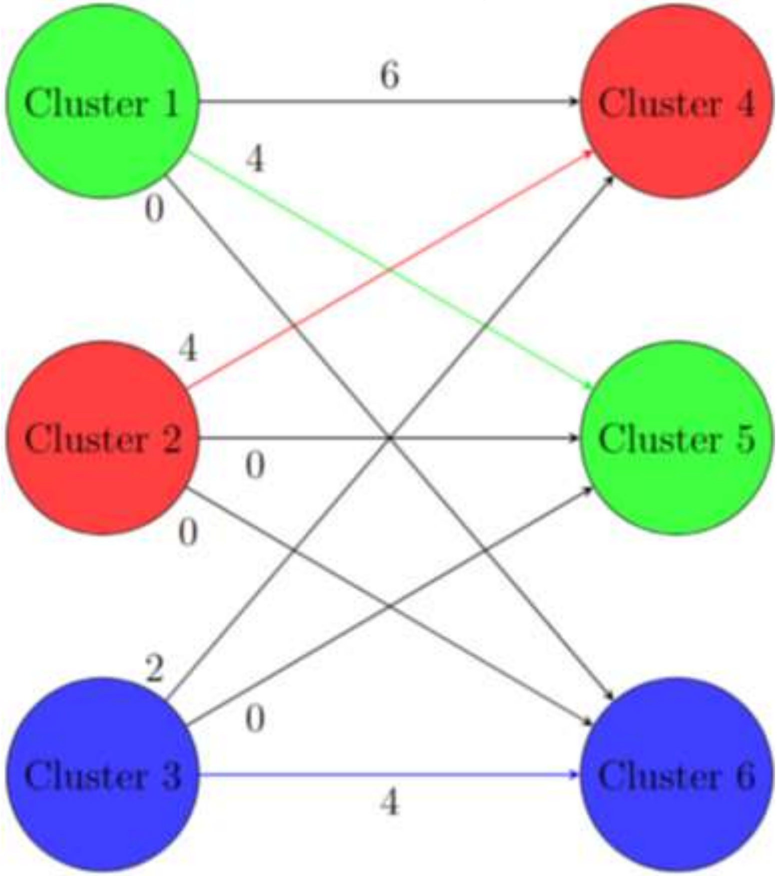


Clusters assignment at a subsequent frame



Maximum Weight Perfect Matching

The bipartite graph's partitions are the clusters at subsequent frames



Edge's weight is the number of nodes shared between clusters

Machine Learning Approach

What it is

Method enabling the binary classification of fragments from specific movements

What it is for

It can identify patterns in the features during the training phase that the algorithmic approach may overlook

Machine Learning Approach

Input

60 samples representing the x, y, z trajectories of the 20-joints for every frame

Feature extraction

Extracted from the distances between joints and the center of mass for every frame

Best features selection

BorderlineSMOTE technique to rebalance the training set

Classification Model

Random Forest Classifier and Leave One Out Cross Validation

Output

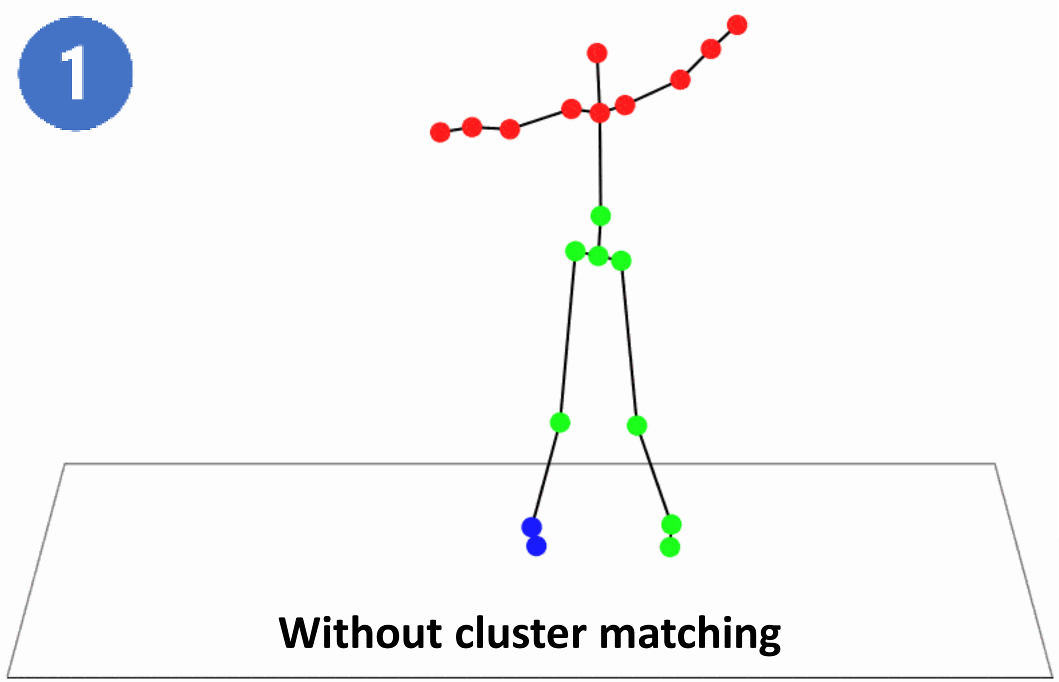
Binary output indicating whether the origin is or is not in the selected class

Algorithmic Approach - Results

	Results	Chance level
Best results **Feature** Angular Momentum **Number of clusters** 4	**On the dataset** 36% accuracy	**At least one correct node over the 20 possible** 19% of chance
	On samples not classified on an extremity of the skeleton 46% accuracy	**At least one correct node over the 16 possible** 24% of chance

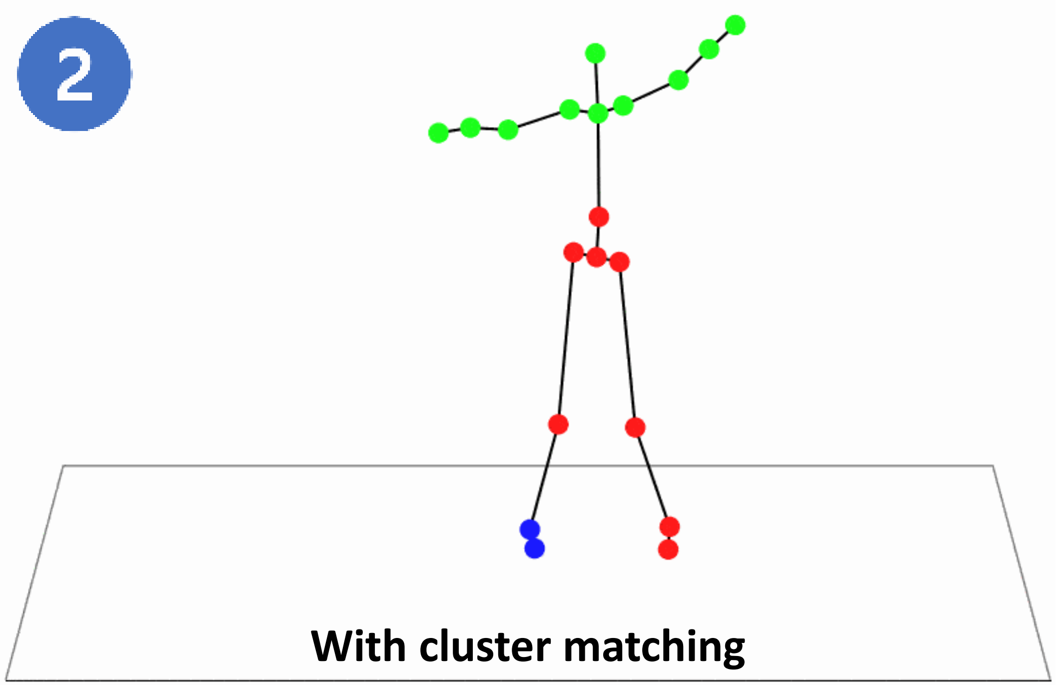
Visualization Approach - Results

1



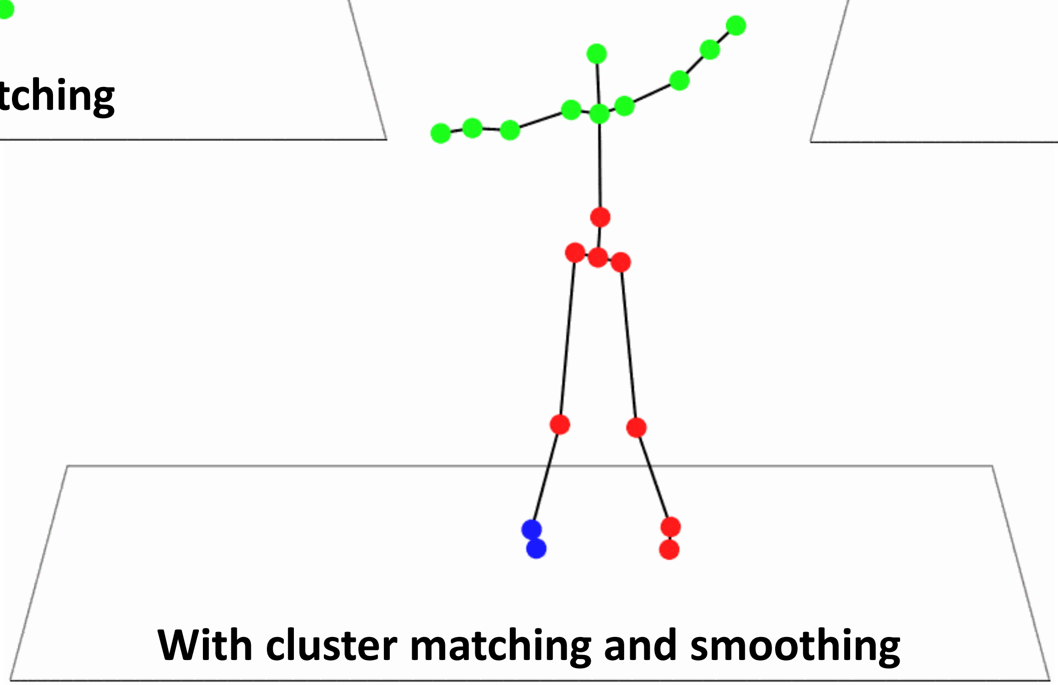
Without cluster matching

2



With cluster matching

3



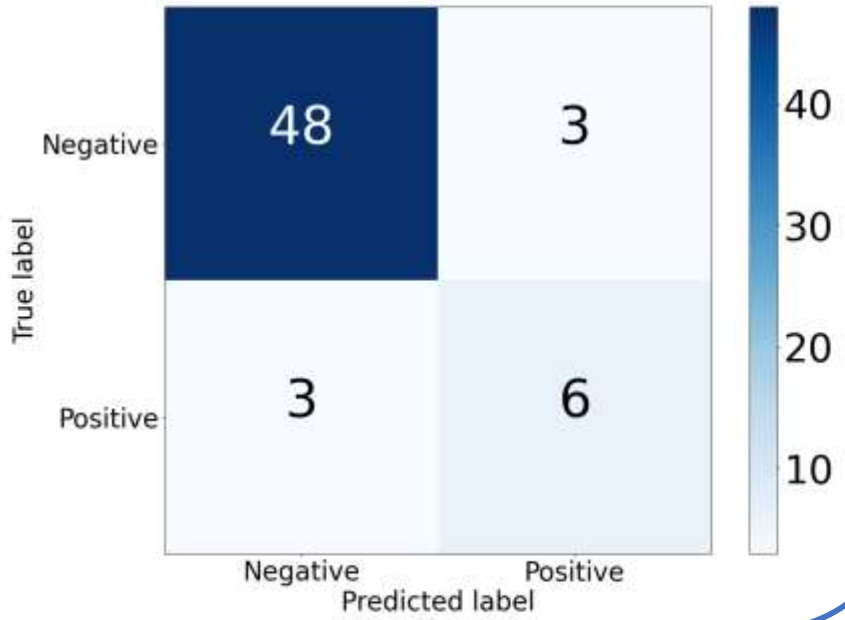
With cluster matching and smoothing

Machine Learning Approach - Results

Small and unbalanced dataset → We couldn't do deep learning neither multiclass classification → We opted for binary classification → Works well on the most frequent classes

Most frequent class confusion matrix

True Skill Statistic: 80%



Conclusions and further researches

Conclusions

Dataset was the main limitation due to the process required to obtain valid samples

Algorithmic and visualization approaches, due to limitations on clusters' creation, cannot classify origins in the borders of the skeleton

Machine Learning approach works well on the most frequent classes

Future researches

With a larger dataset, will be possible multiclassification and deep learning techniques

Reduce the number of constraints in clusters' creation

Multilabel classification on movement fragments characterized by multiple origins

Thank you
for
your attention!

Università
di Genova

Machine Learning – Python Code

```python
import numpy
import scipy.stats

def extractFeatures(raw_feat):
    mean = numpy.mean(raw_feat, axis=0)
    var = numpy.var(raw_feat, axis=0)
    kurt = scipy.stats.kurtosis(raw_feat, axis=0, bias=True)
    skew = scipy.statsskew(raw_feat, axis=0, bias=True)
    corr = numpy.corrcoef(raw_feat, rowvar=False)[numpy.triu_indices(raw_feat,k=1)]
    mad = numpy.mean(numpy.abs(raw_feat - mean), axis=0)
    sem = numpy.std(raw_feat, axis=0) / numpy.sqrt(raw_feat.shape[0])
    energy = numpy.sqrt(numpy.sum(raw_feat ** 2, axis=0))
    iqr = scipy.stats.iqr(raw_feat,axis=0)
    mi = numpy.min(raw_feat, axis=0)
    ma = numpy.max(raw_feat, axis=0)

    fft = numpy.fft.fft(raw_feat, axis=0)
    amplitude_spectrum = numpy.abs(fft)
    frq_info = [numpy.angle(fft)[0, :], # First freq
                numpy.mean(fft.real, axis=0),
                numpy.max(fft.real, axis=0),
                numpy.argmax(fft.real, axis=0),
                numpy.min(fft.real, axis=0),
                numpy.argmin(fft.real, axis=0),
                scipy.stats.skew(amplitude_spectrum, axis=0, bias=True),
                scipy.stats.kurtosis(amplitude_spectrum, axis=0, bias=True) ]

    refined_features = numpy.hstack([mean,var,kurt,skew,corr,mad,sem,energy,iqr,mi,ma,
        numpy.hstack(frq_info)])

    return refined_features
```

```python
from imblearn.over_sampling import BorderlineSMOTE
from sklearn.ensemble import RandomForestClassifier

# Define a function to perform the model training and prediction for one iteration
def train_predict(X, y, i):

    # Remove i-th sample from training
    X_tr = numpy.delete(X, i, axis=0)
    y_tr = numpy.delete(y, i, axis=0)

    # Count labels occurencies
    labels_count = numpy.bincount(y_tr)
    ratio = 0.4

    # Defines the neighborhood of samples to use to generate the synthetic samples.
    k_neigh = numpy.clip(int(labels_count[1] * ratio), 1, len(y))

    # Determine if a minority sample is in "danger"
    m_neigh = numpy.clip(int(labels_count[0] * ratio), 1, len(y))

    # Fit the model with different hyperparameters using BorderlineSMOTE
    bsmote = BorderlineSMOTE(k_neigh, m_neigh)
    X_tr_resampled, y_tr_resampled = bsmote.fit_resample(X_tr, y_tr)

    # Train a first forest
    rf = RandomForestClassifier(n_estimators=500)
    rf.fit(X_tr_resampled,y_tr_resampled)

    # Sort importances in descending order and select the top 50 features
    top_features = numpy.argsort(numpy.array(rf.feature_importances_))[::-1][:50]

    # Select the most important features on train and test set
    X_tr_star_resampled = X_tr_resampled[:, top_features]
    X_t_star_resampled = X[i, top_features].reshape(1, -1)

    # Train a new forest
    rf = RandomForestClassifier(n_estimators=200, max_features=None)
    rf.fit(X_tr_star_resampled, y_tr_resampled)

    # Return prediction on the test instance
    return rf.predict(X_t_star_resampled)[0]

# LOOCV on a dataset
X = ... ; y = ...
predicted_labels = [train_predict(X, y, i) for i in range(len(y))]
```