Engineering Practice Report

Automatic Extraction of Relevant Task Features from Human Demonstrations

15th August 2023

Chaitanya Chawla

Student Number: 03741060
Degree Program: Electrical Engineering and Information Technology
Module: [EI0900] Ingenieurpraxis
Supervisors: Prof Dr. -Ing. Darius Burschka and
M.Sc. Andrei Costinescu

Chair of Robotics, AI, and Realtime Systems
Technical University Munich
School of Computation, Information and Technology

Contents:

Abstract –
Humans, when learning from expert demonstrators, often deduce implicit features of demonstrations which are absent from expert instructions. Such features, although not explicitly stated by the expert, are vital to achieving the task's goal. In this paper, we aim to investigate the presence of various implicit features in day-to-day human tasks and use an oracle-based system to construct a latent representation of these demonstrations. Firstly, we define features to be extracted from the trajectories, such as – intermediate points, surface proximity, speed profiles, and sharp changes in velocity. Subsequently, our system detects the presence of these features in new demonstrations, which are then evaluated and approved by the oracle based on their significance in attaining the desired trajectory. Our findings show that using these features in reconstruction of tasks significantly improves the accuracy trajectory even when features are collected from a single demonstration.

1 Introduction

With the recent increase in deployment of robotic systems across industries, there arises an essential need to develop methods facilitating human to robot knowledge transfer, as well as the robots' subsequent adaptation to new, unseen environments. Robotic Learning, or the process of teaching robots novel skills, including adapting to their dynamic environment, has been of great interest to the scientific community in recent decades. Methods such as – Learning from Demonstration (LfD) and Imitation Learning (*Hussein at al.* **[1]**), are being implemented to teach robots various household skills. Although effective in their approach to achieve diverse task-based goals, most of these methods focus on transferring entire tasks to robots (*Kober & Peters* **[2]**; *Peters et al.* **[3]**). This leads to irrelevant features also being transferred to the robot. Such features, which were not intended to be recorded in the trajectories, often do not play any role in achieving the given task and tend to over-complicate the learned trajectories.

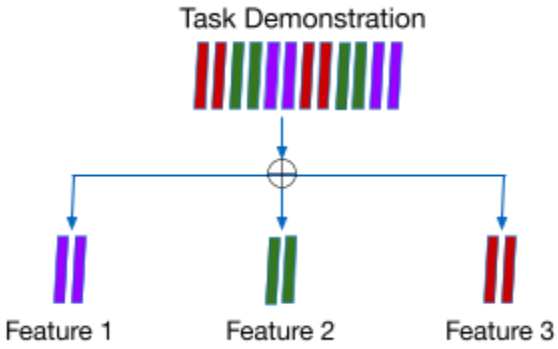

Figure 1: Goal of our work - extracting segments of a trajectory which exhibit certain predefined features

One approach to address this issue is by prioritizing the reconstruction of task features rather than the entire task itself. A robust system should be able to select and omit features from a task representation based on their relevance to goal-attainment and their potential interference with the environment. For example, in the case of filling water from a tap, the system should be able to infer from expert demonstrations that orientation of the glass is irrelevant before filling water and becomes relevant

after filling water. Thus, before filling water, the orientation can be exploited to avoid obstacles in unseen environments. Features such as orientation-control can be detected and utilized to replicate different day-to-day tasks, like painting, transporting, and opening/closing of doors.

Our objective is to extract salient features from a single task and solicit user input to quantify their relevance. As a result, the robot only acquires features relevant for attaining the task at hand, while disregarding the extraneous ones. This methodology ensures that the system respects the intrinsic task characteristics during execution. Furthermore, it also facilitates adaptation to novel environments by enabling the identification and subsequent dismissal of features, interfering with the new environment,
Our contribution includes the development of a system designed to extract diverse, predefined features from a single, visual task demonstration, and incorporating them in a latent representation to significantly improve the task reconstruction.

2 Related Work

2.1 Learning from Demonstration

Recent years have shown that automation of the production cycle has become a major goal for many companies, as robots become more versatile and cheaper, and their use becomes attractive in labor-intensive production chains. Furthermore, as demand for customized products grows, there would be a strong shift from large product batches to small ones (Kumar et al. **[4]**). These two trends combined lead to the demand of robots that learn new tasks quickly and efficiently in order to keep up with the changing production cycle. The traditional form of robot programming stands in the way of this change, as people working in production often lack the background and also the time to implement these changes themselves, and experts must provide the code instead.
Learning skills from demonstrations is appealing, since it allows non-experts to demonstrate solving the target tasks, bypassing the tedium of manually specifying these skills or carefully engineering solutions to the tasks (*Argall et al.* **[5]**). This may be accomplished by simply cloning the original demonstration (*Esmaili et al.* **[6]**), or fitting the demonstration to a trajectory representation (*Kober & Peters* **[2]**; *Peters et al.,* **[3]**) or a policy (*Atkeson et al.,* **[7]**).
While our work also falls into the broad paradigm of LfD, we use it to extract the relevant features rather than the entire trajectory.

2.2 Task Segmentation and Representation

In *Zoliner et al.* **[8]**, a programming by demonstration paradigm was deployed to represent tasks as sequences of action blocks, e.g., [grasp object, move object, .., place object]. Our approach is similar to these in the sense that we also have a pre-defined knowledge base consisting of a set of features.
Conversely, *Garg et. al* **[9]** and *Murali et. al* **[10]** introduce a technique to segment skills from a task by training Convolutional Neural Networks (CNNs) on visual data to learn transition states. *Lee et. al.* **[11]** further proposes a system of task segmentation, which represents tasks as Gaussian Mixture Models (GMMs) and segments using Principal Component Analysis (PCA).

While our system also attempts to segment tasks into subtasks (or skills), we solely do so by utilizing the intermediary points of a task (see section 4.1). Instead of trying to segment trajectory into subtasks, our system strives to understand the physical constraints required by each skill segment, for e.g. orientation, speed, and surface interaction, rather than the skill itself. This approach helps in a finer level of abstraction, affording a better control over the execution of different segments of the trajectory.

In this work, we follow a pre-programmed behavior paradigm, a method typically distinct from conventional Robotic Learning frameworks in that it lacks any engagement with learning structures such as neural layers. Our algorithms detect certain task features based on the feature's experimentally discerned characteristics. Despite its deviation from traditional approaches, this method offers numerous advantages, primarily highlighted by the omission of a learning phase. Consequently, implementation is significantly faster, with reduced computational resources compared to alternative methodologies.

3 Approach

3.1 Dual quaternions

Dual Quaternions (Kenwright **[12]**) are a widely used alternative to the traditional methods of representing translation and rotation such as Euler-Angles, Axis-Angle, and Matrices. Each dual quaternion consists of two quaternions: the real part and the dual part.

$$q = q_r + q_d \cdot \varepsilon \qquad (1)$$

where q_r and q_d are the real and dual quaternions respectively and ε is the dual operator.

We also employed dual quaternion in all our calculations of object trajectories to simplify the computations.

Advantages of using dual quaternions over other representations -

- Singularity-free
- Unified translation and rotation representation in a single state.
- Simplified mathematical operations
- Most efficient and compact representation

Due to their more accessible and understandable usage, we have used dual quaternions throughout our work.

3.2 Detecting Features from Demonstrations

The project is divided into three main feature sets - (i) Intermediary point detection, (ii) Surface proximity, and (iii) Velocity Features. All three features were executed on prerecorded trajectories.

3.2.1 Intermediary Point Detection – Octree-Based Point Cloud Segmentation

In recent years, Octree structures have been used in compression (Garcia et al. **[13]**) and reconstruction of point clouds (Gao et. al **[14]**). We record a manipulated object's trajectory using AprilTags (Olson et. al **[15]**). We then employ octrees to detect interruptions in the object's trajectory, indicating moments when the demonstrator halted the manipulation.

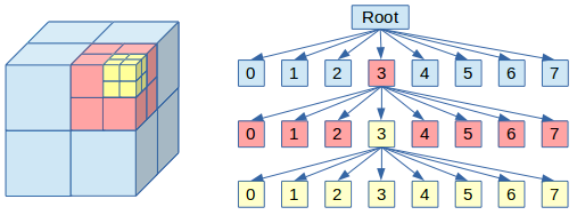

Figure 2: Binning of point clouds in Octrees and Voxel Grids

Such a trait is common in tasks such as scanning an object at a supermarket checkout or filling water from a tap. This process consequently also helps in dividing the trajectory into smaller sub-trajectories, with these intermediary points acting as breakpoints. These sub-trajectories are then further analyzed for features such as the surface proximity and object velocity.

3.2.2 Surface Proximity -

The next feature described here aims to achieve the detection of a class of tasks where the manipulated object stays in close proximity to a surface in the environment. For example - wiping a surface, writing on the blackboard, or painting a wall.

This approach involved a three-stage process. Initially, we record the object's trajectory, executed by the demonstrator. Then, we capture the point cloud data of the surfaces present in the scene using Costinescu et. al **[16]** and conduct an offline computation to determine the two-dimensional limits of the surfaces in the scene. These limits were then compared with the object's trajectory to assess its proximity to the surfaces.

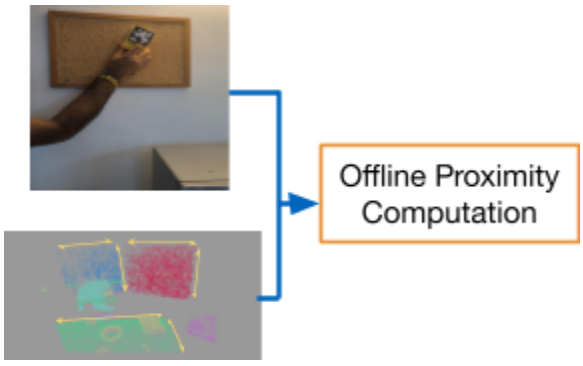

Figure 3: Workflow of Surface Proximity Pipeline

3.2.3 Velocity Features -

Finally, we also designed a set of features to analyze the velocity profile of a trajectory. Humans, unknowingly, exhibit certain velocity characteristics to aptly achieve goals of tasks. Instances such as carrying a glass full of water, or writing on paper with a pen exemplify our efforts in such scenarios to minimize speeds and maintain consistent object orientations to achieve the given goal. To detect and include these intrinsic features in our representation, we set up a pipeline to detect five key features – 'speed-less-than', 'constant speed', 'constant angular velocity', 'accelerating/decelerating speed', and 'constant direction'. A kinematic analysis is conducted on the manipulated object's trajectory to detect different velocity features (e.g. when the object maintains a constant speed) and experiment with parameters (e.g. threshold for constant speed) to accurately detect the aforementioned features.

4 Methodology and Implementation

4.1 Intermediary Point detection

This feature is implemented by first capturing the 3d poses of the manipulated object throughout its trajectory and then converting it into a dense point cloud representation. Following this, leveraging an Octree structure (see Fig. 4), the entire point cloud is segmented by binning into voxel grids of predefined sizes. Each voxel within the grid represents a single sample or data point on a regularly spaced 3D grid.

Depending on the number of points contained in a voxel, a voxel is classified as an intermediate point. Through empirical

experimentation, the threshold number of points in a voxel was determined to be approximately 100 points per voxel of side 5 cm. Furthermore, we also enforce a minimum separation distance between two identified intermediary points to prevent a single point from being misclassified as two separate intermediary points. Finally, the oracle validates the intended detections among the identified points.

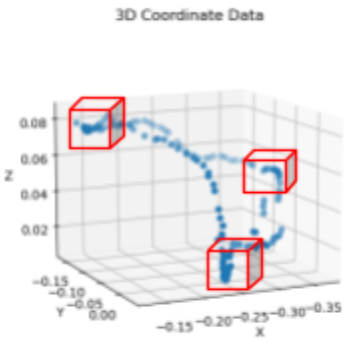

Figure 4: Detected Intermediary points from a trajectory

One evident fallback with this approach is the variability of the threshold number of points within a voxel grid, depending on the task at hand. As the duration of the pause at an intermediary point increases, the density of the point cloud within the voxel also increases correspondingly.

4.2 Surface Proximity

With the help of **[16]**, we implement the detection and rendering of all planar surfaces from a frame in the form of a point cloud. We further perform a Singular Value Decomposition - SVD (Wall et. al **[17]**) upon the point cloud data to determine the value of eigenvectors and the directions in which each surface extends. Firstly, all data points are converted from the camera to the ground reference frame. Then, through SVD (see equation **2**), we obtain the three matrices *U*, *Σ*, and *V*.

$$M = U \Sigma V^* \quad (2)$$

$$\Sigma = diag(\sigma_1, \sigma_2, \ldots \sigma_n) \quad (3)$$

$$M M^* u_i = \sigma_i^2 u_i \quad (4)$$

$$M^* M v_i = \sigma_i^2 v_i \quad (5)$$

where *M* is the point cloud data with respect to ground frame and *M** denotes its conjugate transpose, *U* represents eigenvectors of *MM**, *Σ* is a rectangular diagonal matrix containing the singular values of *M*, and *V* represents eigenvectors of the matrix *M*M*. The eigenvectors form an orthonormal basis for M.

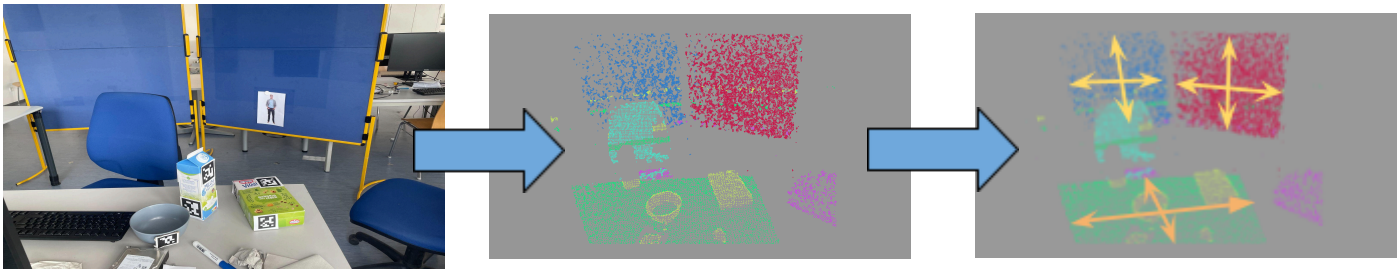

Figure 5 : Point cloud representation of the detected surfaces along with their eigenvectors

As $M \in \mathbb{R}^{3xn}$ in our case, we take columns of $V \in \mathbb{R}^{3x3}$ as the coordinate axis of our 3 dimensional space. Here, we assume that the surface with the maximum number of visible points is the primary surface, on which the task is being executed. Finally, the object location is compared with the surface's 3D limits and determines whether the object was in close proximity. Figure **5** shows the surfaces along with their eigenvectors.

4.3 Velocity Features

In the final section of our work, we implemented 5 features relating to the velocity of the manipulated object –

I. **Speed less than** - Given a predefined threshold speed, the pipeline returns intervals of the trajectory where the speed falls below this threshold. Carrying a plate full of food is one such instance where it is necessary to keep the speeds below a certain value.

II. **Constant speed** - This section aims to identify segments in a trajectory, where the object maintains a constant speed. Due to the challenge of manually establishing a fixed percentage change in speed to denote constancy, we afford the oracle discretion to specify the percentage change for different intervals. This approach provides the oracle freedom to define the concept of "constant" speed based on individual interpretation. Employing a moving average of the object's raw speeds, we mitigate the influence of outliers on the detected segments.

III. **Constant angular velocity** - Similar to the methods in Section **4.3 II**, the detection of limits for constant angular velocity is also based on the oracle's choice of the parameters for different intervals. We discern the presence of the feature on the basis of two criteria. Firstly, whether the axis of rotation is being kept constant (see **eqn. [7]**). Secondly, whether the rotation about an axis is being kept constant (see **eqn. [8]**). When both criteria fulfill specific threshold values, then we say that the object maintains a constant angular velocity.

$$\Delta\mathbf{rot} = q_{t1,r}^{-1} * q_{t2,r} \quad (6)$$

$$\Delta\mathbf{ax} = quat2ax(q_{t2,r}) - quat2ax(q_{t1,r}) \quad (7)$$

$$avg\_rot = quat2angle(\Delta\mathbf{rot}) * quat2ax(q_{t2,r}) / (t2 - t1) \quad (8)$$

where $\mathbf{q_{t1,r}}$ and $\mathbf{q_{t2,r}}$ denote the rotation quaternion of the dual quaternions at time **t1** and **t2** respectively. Δ**ax** represents the change in axis of rotation. **quat2angle()** and **quat2ax()** are functions which convert quaternion notations to axis-angle notation and return the angle and axis of the quaternion respectively.

IV. **Increasing/Decreasing Speed** - This feature reports the intervals in which the object is being accelerated or decelerated.

V. **Constant Direction** - This feature seeks to segment instances of the trajectory where the object exhibits a constant direction. This feature is mainly developed to detect instances of shaking motion exhibited by the demonstrator, characterized by continuous changes in the trajectory's direction.

The workflow of the first three velocity features depends on parameters, which have been experimentally determined to fulfill the requirements of the tested tasks.

5 Experiments and Results

5.1 Quantitative Results

5.1.1 Detecting intermediary points

To test the accuracy of our pipeline, we recorded 10 trajectories from different users by asking them to intentionally include intermediary points lasting longer than 3 seconds. Table 1 shows the results in the form of a confusion matrix.

		Actual Values	
		+ve	-ve
Predicted Values	**+ve**	**27**	**5**
	-ve	**7**	**-**

Table 1: Confusion matrix for detection of 34 intermediary points present in 10 trajectories

This shows a precision of 84.3% and an accuracy of 79.4%. The oracle further is queried for the correctness of the detected points to obtain the true intermediary points of the trajectory.

5.1.2 Surface proximity detection

We also test our surface proximity detection module with 10 demonstrations containing 32 time intervals where the object is in close proximity to a surface. The results are shown in table 2. This showed a high accuracy of 90.6%. In a similar manner to the previous section, the predicted object-surface proximity time intervals are queried to the oracle to confirm their correctness.

		Actual Values	
		+ve	-ve
Predicted Values	**+ve**	**29**	**-**
	-ve	**3**	**-**

Table 2: Confusion matrix for detection of 32 time intervals consisting of object-surface proximities

Note that for each of the above two results, we pre-programmed a threshold to decide how long an object needs to stay at its position to be considered an intermediary point and how close the object needs to be to the surface for it to be considered in its proximity. The above-mentioned accuracies could vary depending on the task and the threshold set for the specific task by the user.

5.1.3 Velocity profile segmentation

Finally, we conducted similar experiments for the 5 segments of the velocity profile.

		Actual Values	
		+ve	-ve
Predicted Values	**+ve**	**10**	**-**
	-ve	**2**	**-**

Table 3: Confusion matrix for speed-less-than detection from 12 true values

	Actual Values		
Predicted Values		**+ve**	**-ve**
	+ve	**7**	**-**
	-ve	**4**	**-**

Table 4: Confusion matrix for constant-speed detection from 11 true values

	Actual Values		
Predicted Values		**+ve**	**-ve**
	+ve	**4**	**5**
	-ve	**5**	**-**

Table 5: Confusion matrix for accelerating/decelerating-speed detection from 9 true values

	Actual Values		
Predicted Values		**+ve**	**-ve**
	+ve	**6**	**-**
	-ve	**5**	**-**

Table 6: Confusion matrix for constant-angular-velocity detections from 11 true values

The lowest accuracy of 44.4% was observed in the accelerating/decelerating speed section, whereas the highest of 83.3% was observed in the speed-less-than section. The constant-speed section and constant-angular-velocity section showed accuracies of 63.63% and 54.54% respectively. As the last 3 features of velocity profiles show very less accuracies (avg. 54%), we further test them out with simulated trajectories.

It is important to note that throughout our work the term “constant” pertains to features which remain constant for a fixed duration of time, rather than for the entirety of the trajectory. This duration was experimentally determined to be ~3 seconds for most features.

5.2 Simulated Dataset

We develop a simulated dataset generator. The user can specify start pose, and time intervals with constant velocity, constant acceleration, and constant angular velocity.
The velocity features on the simulated dataset show a 100% confirming the correctness of our approach. This means that velocity profiles are tough to judge by humans and thus recreation of such discrete velocity features that can be detected is also a difficult task. Accelerations display the worst results as they can not be very well segmented.

5.3 Limitations

Our approach provides satisfactory results for detection of various skills, when the tasks that it observes have features falling within the predefined thresholds of the task.
However, when analyzing a demonstrated task the difficulty consists firstly in accounting for the large variability that may exist between demonstrations and deciding what features of the motion should be reproduced (extracting task constraints).

One of the biggest disadvantages of this approach is that the characteristics are defined manually through experiments. This

could potentially lead to false parametrization, as a single set of parameters would not be able to define the feature in a very general way. So, the selected parameters would provide accurate feature detection, only for a limited amount of variation in the task. For example, in the case of intermediary point detection, it was found to be difficult to set a universal point per voxel threshold to classify all intermediary points in all kinds of tasks. Furthermore, also as an oracle, it is tough to determine appropriate values for these parameters as in the case of constant-speed detections. They could vary for different tasks, and thus make it tough to assign a single value which would be able to correctly classify all instances of a feature in various tasks.

6 Conclusion and Future Work

To summarize, through our work we were able to segment parts of the trajectory containing intermediary points and object-surface proximity quite well. Whereas there is still room for improvement in detection of different velocity profiles.

To further improve our system, we propose some developments which could make the system more accessible. To begin with, some features such as the surface proximity detection can be implemented in real time to provide live feedback to the user. Furthermore, we could optimize the parameters for detection of various features. At the moment, they were set experimentally and can vary depending on task and the demonstrator. Finally, we would like to incorporate these features with

REFERENCES

[1] Hussein, A., Gaber, M. M., Elyan, E., & Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, *50*(2), 1-35.

[2] Kober, J., & Peters, J. (2009, May). Learning motor primitives for robotics. In *2009 IEEE International Conference on Robotics and Automation* (pp. 2112-2118). IEEE.

[3] Peters, J., Kober, J., Mülling, K., Krämer, O., & Neumann, G. (2013). Towards robot skill learning: From simple skills to table tennis. In *Machine Learning and Knowledge Discovery in Databases: ECML PKDD 2013*.

[4] Kumar A., From mass customization to mass personalization: A strategic transformation, Int. J. Flexible Manuf. Syst. 19 (4) (2007) 533–547.

[5] Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, *57*(5), 469-483.

[6] Esmaili, N., Sammut, C., & Shirazi, G. M. (1995, October). Behavioural cloning in control of a dynamic system. In *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century* (Vol. 3, pp. 2904-2909). IEEE.

[7] Atkeson, C. G., & Schaal, S. (1997, July). Robot learning from demonstration. In *ICML* (Vol. 97, pp. 12-20).

[8] Zoliner, R., Pardowitz, M., Knoop, S., & Dillmann, R. (2005, April). Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration. In *Proceedings of the 2005 IEEE International Conference On Robotics and Automation* (pp. 1535-1540). IEEE.

[9] Murali, A., Garg, A., Krishnan, S., Pokorny, F. T., Abbeel, P., Darrell, T., & Goldberg, K. (2016, May). Tsc-dl: Unsupervised trajectory

segmentation of multi-modal surgical demonstrations with deep learning. In *2016 IEEE international conference on robotics and automation (ICRA)* (pp. 4150-4157). IEEE.

[10] Garg, A., Krishnan, S., Murali, A., Pokorny, F. T., Abbeel, P., Darrell, T., & Goldberg, K. (2015). In *NIPS Workshop on Feature Extraction*.

[11] Lee, S. H., Suh, I. H., Calinon, S., & Johansson, R. (2015). Autonomous framework for segmenting robot trajectories of manipulation task. *Autonomous robots*, *38*, 107-141.

[12] Kenwright, B. (2012). A beginners guide to dual-quaternions: what they are, how they work, and how to use them for 3D character hierarchies.

[13] Garcia, Diogo C., et al. "Geometry coding for dynamic voxelized point clouds using octrees and multiple contexts." *IEEE Transactions on Image Processing* 29 (2019): 313-322.

[14] Gao, X., Shen, H., & Panozzo, D. (2019, August). Feature preserving octree-based hexahedral meshing. In Computer graphics forum (Vol. 38, No. 5, pp. 135-149).

[15] Olson, E. (2011, May). AprilTag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation* (pp. 3400-3407). IEEE.

[16] Andrei Costinescu, Andrei Utils, (2023), Github Repository https://bitbucket.org/andreicostinescu/andreiutils/src/main/

[17] Wall, M. E., Rechtsteiner, A., & Rocha, L. M. (2003). Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis* (pp. 91-109). Boston, MA: Springer US.