MECA-H-419 - Data-Driven Engineering

Electromechanical Engineering - Robotics section

Project report

Study of the bouncing motion of a ball using Computer Vision and Data-Driven methods

Authors:

Mohammadjavad Rahimi Dolatabad
Louise Massager
Dinh-Hao Nguyen
Yiming Yuan

Professors:

Emanuele Garone
Miguel Alfonso Mendez
Alessandro Parente
Mehrdad Teratani

June 11, 2022

Academic year 2021-2022

External Links

All the codes related to this project can be found in the following GitHub repository:

`https://github.com/LouiseMassager/DDE_project_bouncingball`

Contents

1 | Introduction

1.1 Motivation

As of today, data-intensive science is used frequently in most fields. It is especially true for fluid mechanics and computer science. Not so long ago, scientific research had a hard time to connect experimental data, theoretical models, and simulation results together. The breakthrough of data-driven research has enabled a reconciliation of theory, experiments and simulations. New theories in physics have emerged purely from data-driven processes. It also comes with the advantage of being completely free of human bias contrary to experimental and theoretical formulas; however, there is a limitation of data-driven engineering. The data is for example limited to the technology that generates it. Furthermore, while the process is free of human bias, the data must still be interpreted, which requires a certain context and existing knowledge. Machine learning, on the other hand, can be used for a multitude of goals. Such works mainly focus on its use to:

- Gather and process experimental data (cf. sections 2 and 3)
- Use an existing model to extract information from the experimental data (cf. section 4)
- Build a statistical model for labelling or prediction (cf sections 6 and 7)

This report focus on a physics-based problem and its tracking using computer science. This choice comes from the fact that computer vision is used considerably in Robotics engineering and is mostly to track robots that are controlled using the knowledge of several theoretical formulas expressing the physics behind their movement.

1.2 Objectives

The project revolves around the identification of characteristics around a bouncing ball released at a certain height with or without initial velocity. Computer vision has been used throughout this project in order to acquire experimental data. The trajectory of the ball are determined based on the recording of the ball bouncing on the floor and the analysis of the recordings. The experimental data must then be processed. Afterwards, data-driven process can be used in order to extract characteristics of the experimentation from the data as well as predict the trajectory for lacking data. The project has accordingly be divided in the following several objectives:

- Extract experimental data on the trajectory of the ball using computer vision to track the ball from recordings of the experiences.
- Process the experimental data by removing the outliers and using clustering to characterise each bounces of the ball.
- Use a Least-Square optimisation to deduce the initial height of the ball and the coefficient of restitution from a known physical equation of the phenomenon.
- Predict the ball's future trajectory from previous positions of the ball using a linear regression.
- Deduce the type of ball used from the acquired data using PCA.
- Deduce the type of ball based on its provided features with a neural network.

2 | Tracking the ball with computer vision

In this project, computer vision has been utilized to acquire the data on which further development will be realized. The tracking has been realized with the python package of *opencv*. In both cases, the data-set extracted from the tracking is composed of 3 features: the time of the frame with the horizontal and vertical distances of the ball (in pixels) at this time.

2.1 Ball tracking based on its shape

A first, tracking was realised based on the shape of the ball. The idea behind it was to use Hough Circle Transform to detect circles in all frames. By entering the radius of the ball and few parameters to tweak the sensitivity of the circle detection, great results were found when the ball was not moving too fast. It was first tested in real time by moving the ball in front of the webcam. It was then tested with a pendulum and already started to show less good results when the ball was launched from high angles. Once the tracking was used on videos of the bouncing ball experiment, the ball could not be tracked most of the time. It was then apparent that the camera no longer detected a round shape but rather a blurry ovular shape when the ball moved at too high speed. As shown in Figure 2.1, this was the case when the ball was released in the bouncing ball experiment but not in the case of the pendulum. This kind of tracking could therefore not been used for this application. Other application, such as the tracking of a pendulum, could however benefit from this type of tracking.

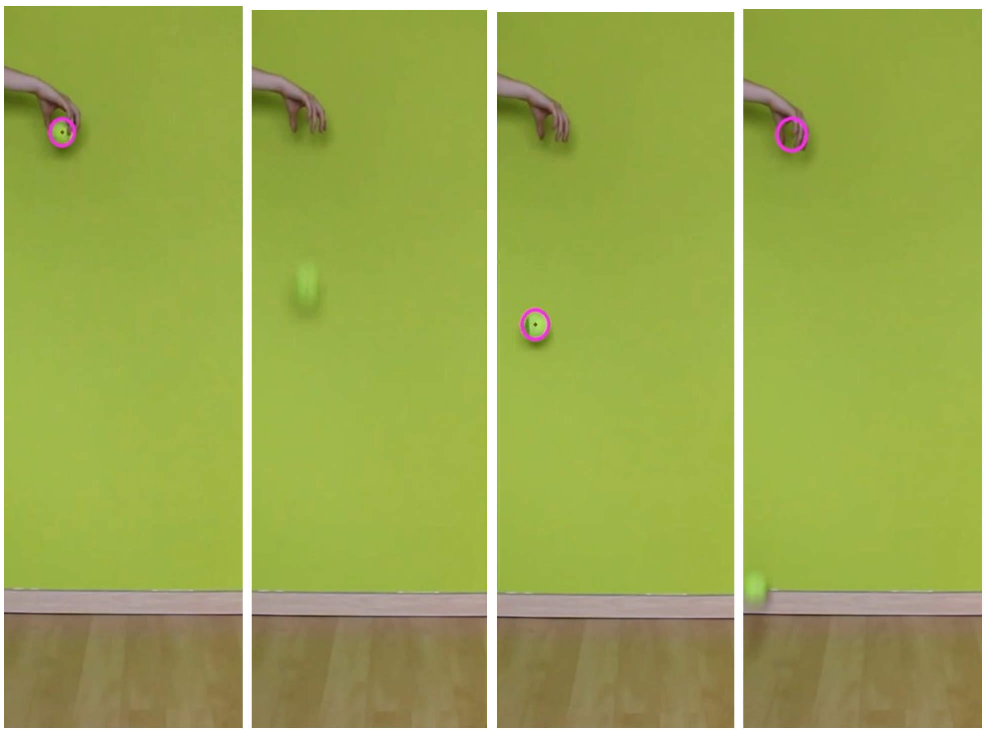


Figure 2.1: Tracking of the center of the ball's position based on the round shape of the ball for the data-set *ball_1_3*.

2.2 Ball tracking based on its color

In the end, the ball was tracked based on its color. It proved to be a highly efficient tracking as the exact position of the ball could be determined at any instances, contrary to the tracking based on its shape. Although the ball seemed quite deformed and blurry most of the time, the position of its center could still be detected quite accurately. In Figure 2.2, the deformation and blurriness of the ball when falling can be noticed as well as the rather accurate tracking. It is also interesting to notice that, although the wall was in itself of similar color as the ball (both in green), the tracking was still highly efficient. It is probably due to the fact that the tracking is based on the color in a HSV format instead of the traditional RGB that is better distinguished by the human eye.

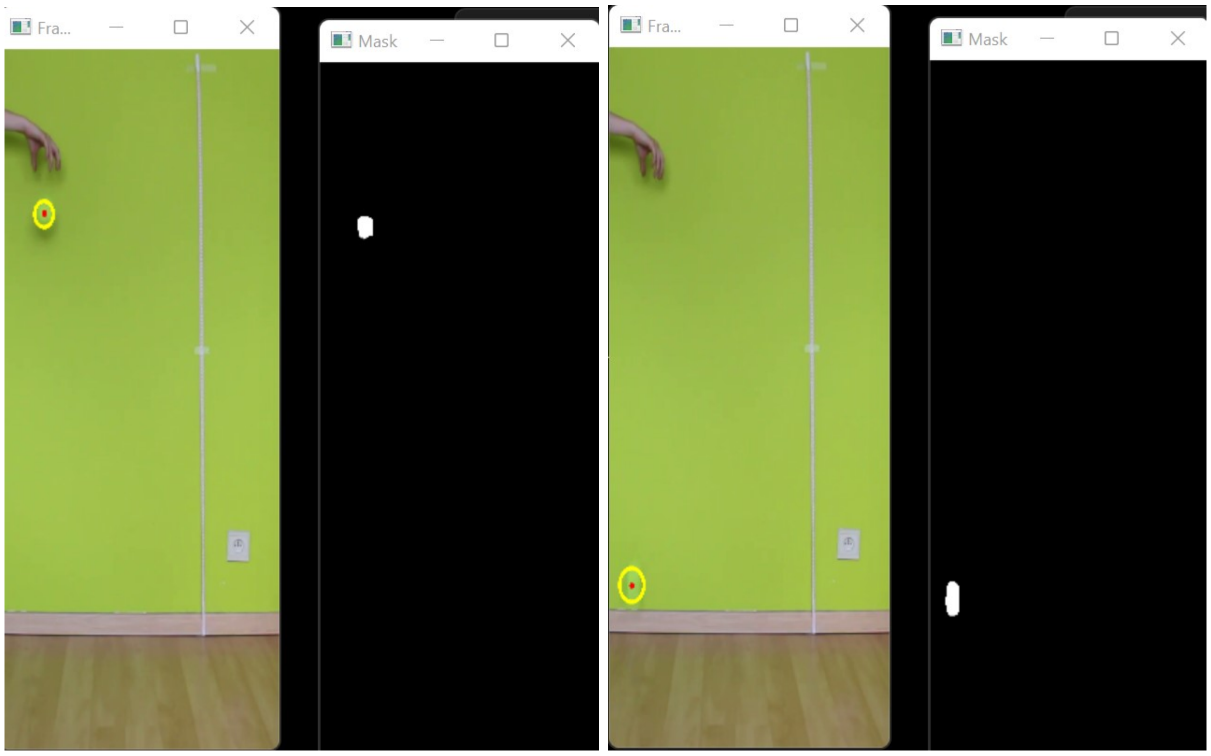


Figure 2.2: Tracking of the center of the ball's position based on the color of the tennis ball for the data-set *ball_ 1_ 3*.

In some videos, the presence of some outliers were noticed as the ball was from time to time detected at the level of the hand releasing the ball or the tape on the wall. The ball also moved slightly towards the camera during most experimentation. A treatment of the data had therefore to be performed in order to efficiently remove outliers and gather the data corresponding to each rebounds using clustering. This clustering enabled the removal of the offset caused by the ball coming towards the camera.

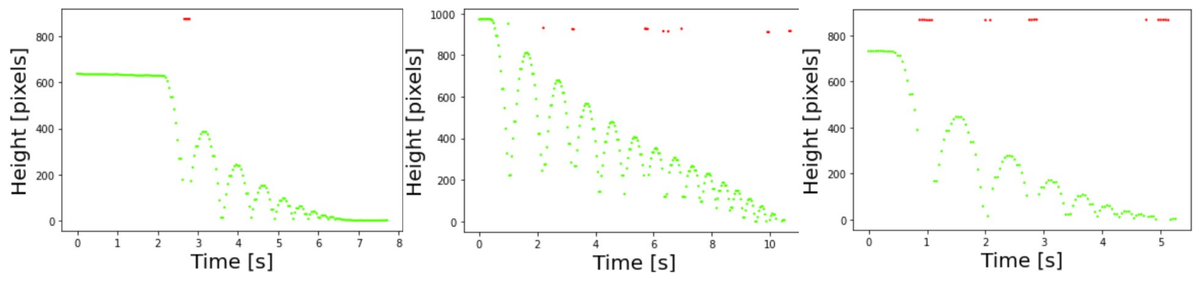
3 | Treatment of the experimental data

3.1 Outlier detection

The tracking of the ball brought mostly reliable data. However, the tracking was sometimes not successful at each instances. A ball was from time to time detected on the wall due to the presence of tape or even due to the shadows of the hand releasing the ball initially. Those instances consisted in outliers in the data. In order to remove them, a short code was generated in Python. Special libraries from *sklearn* have been used for this purpose.

The code has been tested on 3 videos where the presence of outliers was sufficiently noticeable to be considered influential enough to affect further results if not removed. The three videos are referred as *"ball_1_4"*, *"ball_1_6"* and *"ball_2_2"*. 2 types of outliers detection were compared: *LocalOutlierFactor* and *EllipticEnvelope*.

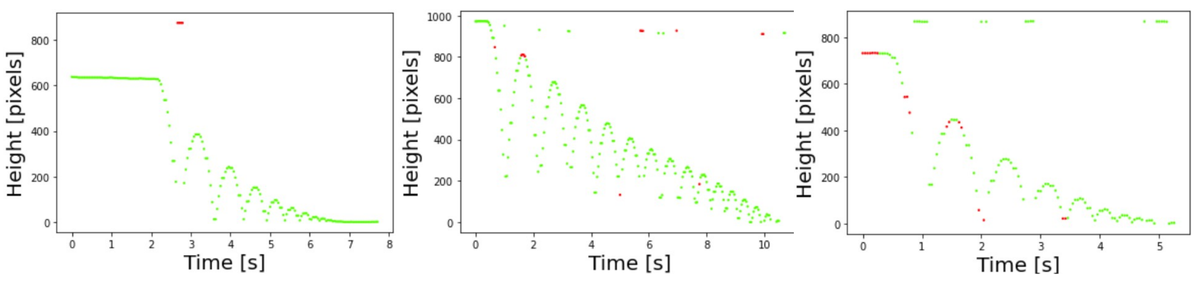
Both methods rely on the knowledge of the contamination level representing a rough estimation of the quantity of outliers present in the data. This parameter, common to both methods, has been chosen by modifying it until the seemingly best result was obtained. This lead to the graphs present in Figures 3.1 and 3.2.



(a) Contamination of 0.02. (b) Contamination of 0.05. (c) Contamination of 0.14.

Figure 3.1: Graphs of the outlier detection with ***EllipticEnvelope*** for the data-sets *ball_1_4*, *ball_2_2* and *ball_1_6* of contamination levels 0.02, 0.05 and 0.14 respectively. The outliers are represented in red while the remaining data is in green.

One common way of performing outlier detection is to assume that the data should fit a known distribution, for instance a Gaussian distribution. From this assumption, a "shape" to the data can be imposed and outlying observations detected in points far away from this imposed shape. The object *covariance.EllipticEnvelope*, provided by *scikit-learn*, has been used to for this purpose by fitting an ellipse to the centre of the data points, ignoring points outside the central mode. This outlier detection performed perfectly as all outliers in the 3 used data-sets have been verified to be detected at the exception of one point at $\pm$1s in *ball_2_2* (cf. Figure 3.1b).



(a) Contamination of 0.02. (b) Contamination of 0.05. (c) Contamination of 0.14.

Figure 3.2: Graphs of the outlier detection with ***LocalOutlierFactor*** for the data-sets *ball_1_4*, *ball_2_2* and *ball_1_6* of contamination levels 0.02, 0.05 and 0.14 respectively. The outliers are represented in red while the remaining data is in green.

LocalOutlierFactor was also tested in a first time but lead to inconclusive results. This method consists of an unsupervised Outlier Detection using the Local Outlier Factor (LOF). The locality is given by k-nearest neighbors, whose distance is used to estimate the local density. By comparing the local density of a sample to the local densities of its neighbors, one can identify samples that have a substantially lower density than their neighbors. These are considered outliers. This method worked great when few outliers were present (cf. *"ball_1_4"*) but falsely detected outliers in the case were outliers appeared several times throughout the experience (contamination level above 0.02). This method to perform outlier detection is usually more efficient on moderately high dimensional data-sets. Our data-set being only of 3 dimensions (time, height and horizontal distance), the *EllipticEnvelope* was preferred.

3.2 Clustering of the ball bounces

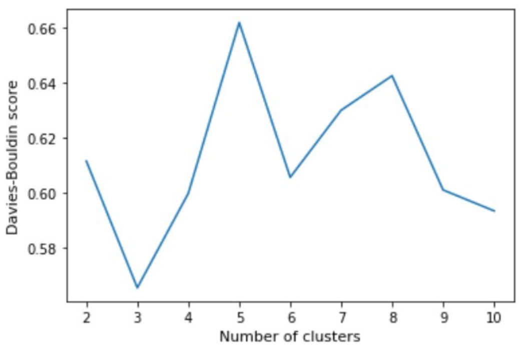
From a 2D video, only the vertical and horizontal movements of the ball can be extracted. However, in reality, the ball can also move towards or away from the camera. Observed from the camera, this can be perceived as a variation in altitude of the contact point between the ball and the ground. The main idea behind the clustering is therefore to subdivide the data in the number of bounces in order to adjust the ground reference for each of them individually.

Clustering the data would indeed allow to associate to a given position and time the number of bounces the ball has already made. Thus, the ground level can be adjusted by taking as local ground reference the minimum altitude within a cluster.

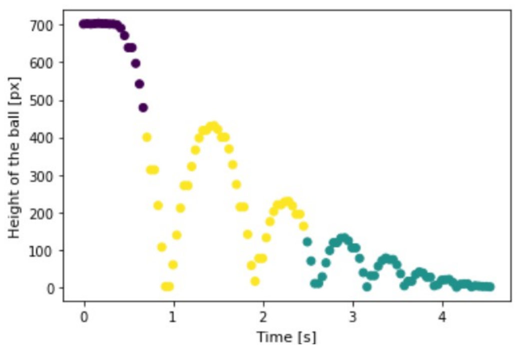
3.2.1 Time and height features

K-Means

After centering and scaling the data, K-Means was used. To automatically select the correct number of clusters, the Davies-Bouldin score (DBS) is used. It indicates how well the clustering has been performed. In fact, it represents the average "similarity" of clusters, where similarity is a measure that relates the distance of clusters to their size. A model with a lower DBS has a better separation between the clusters. Therefore, the optimal number of clusters is the one for which the DBS is minimal.



(a) Davies-Bouldin score.



(b) K-Means clustering.

Figure 3.3: Clustering of the ball's bounces with time and height features by K-Means method applied on the data-set *ball_1_3*.

K-Means minimizes the variances within the clusters (i.e. the squared Euclidean distances). It is very suitable for cloud data, but not our case since the points of each bounce are rather close to each other in both time and height.

VQPCA

Since K-Means was not appropriate for this type of dataset, the VQPCA was used as it was better for non-cloud data. The VQPCA algorithm was also tested, but the goal of isolating each bounce was still not achieved. This is because the clustering was in a first time applied with time and height as features. Height is however not a suited choice of feature as for each bounce, points of the same height would be reached. Height is therefore not a feature proper to each bounce and shouldn't be used to differentiate them.

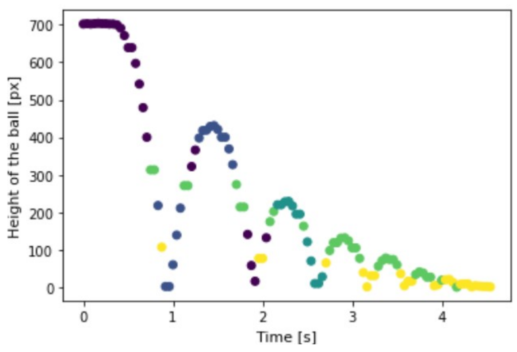


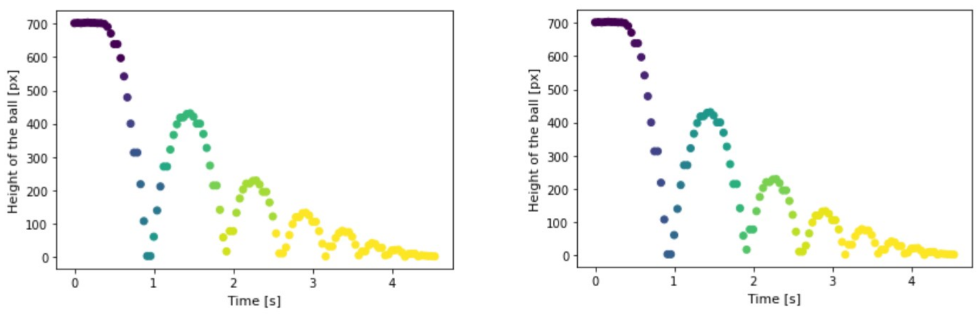
Figure 3.4: Clustering of the ball's bounces with time and height features by VQPCA method applied on the data-set *ball_1_3*.

DBSCAN

DBSCAN is another clustering method. It stands for "Density-Based Spatial Clustering of Applications with Noise" and is able to find arbitrary shaped clusters and clusters with noise (i.e. outliers). The main idea behind DBSCAN is that a point belongs to a cluster if it is close to many points from that cluster.

There are two key parameters of DBSCAN:

- **eps**: The distance that specifies the neighborhoods. Two points are considered to be neighbors if the distance between them are less than or equal to eps.
- **min_samples**: Minimum number of data points to define a cluster



```
Number of clusters = 16
```

(a) eps=0.2 and min_samples=1.

```
Number of clusters = 31
```

(b) eps=0.14 and min_samples=1.

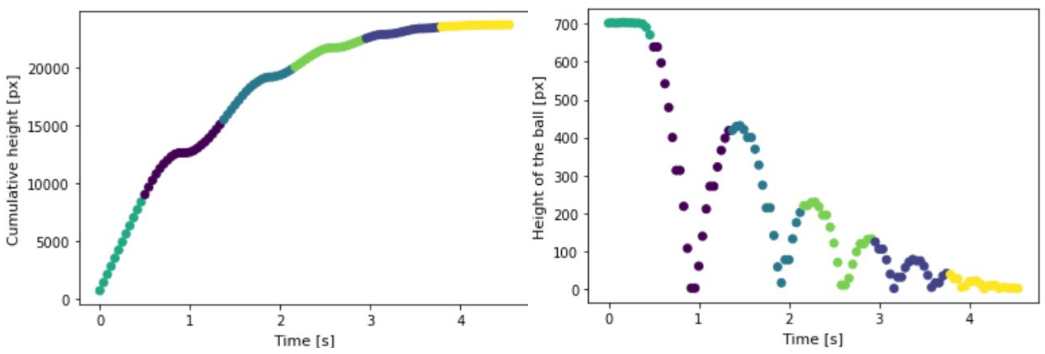
Figure 3.5: Clustering of the ball's bounces with time and height features by DBSCAN method applied on the data-set *ball_ 1_ 3*.

The clusters obtained with DBSCAN using both time and ball height are close to what we wanted. However, the number of clusters is too large and some contain only one or very few points. An algorithm can be written to distribute these small clusters into the larger ones, but this would distort the automatic aspect of the DBSCAN algorithm.

3.2.2 Time and accumulative height features

K-Means

Previously, we have seen that K-Means tends to put in the same cluster the data that have the same ball height. To overcome this problem, a cluster based on accumulative height has been performed. The term *accumulative* refers in this context to the sum of the current value and all the previous values in time. This way, similar features cannot be detected between different bounces anymore as before with the use of the height. The results of the K-Means clustering with those features are however still not very interesting to differentiate the bounces (cf. Figure 3.6).



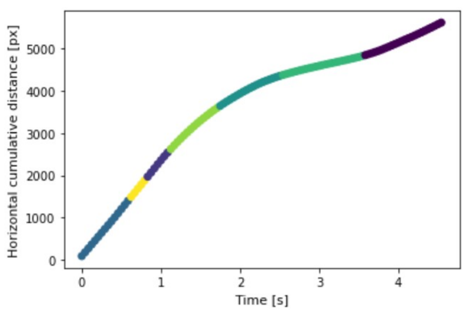
(a) Accumulative height over time. (b) Height over time.

Figure 3.6: Clustering of the ball's bounces with time and accumulative height features by K-Means method applied on the data-set *ball_ 1_ 3*.

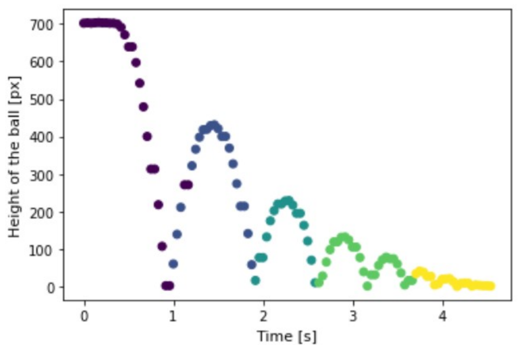
3.2.3 Time, height and accumulative horizontal distance features

VQPCA

Lastly, the expected results were the closest with the inclusion of the accumulative horizontal distance applied on VQPCA. Only 5 clusters could be imposed whatsoever, leading to the merging of the last bounces together. Few points belonging in reality to the second bounce were also included in the cluster for the first bounce as can be observed in Figure 3.7.



(a) Accumulative horizontal distance over time.



(b) Height over time.

Figure 3.7: Clustering of the ball's bounces with time, height and accumulative horizontal distance features applied with VQPCA on the data-set *ball_ 1_ 3*.

3.2.4 Interpretation of the results

It is important to note that the clustering based on accumulative height and horizontal distance has been performed with VQPCA, DBSCAN and K-Means but lead to inconclusive results in the cases not presented. Furthermore, clustering based on other features such as velocity and acceleration have also been tested but didn't help split the data appropriately for the same reasons as for the clustering based on the height, namely similar features between different bounces.

While not optimal, the last clustering proposed allowed for a more or less clear division of the bounces from a top reference. In order to improve the results, it is highly hinted than another method that clustering should be used as the bounces of the ball are only clearly distinguishable based on the time. A better splitting method should rely more on the similarity of the data to separate it by forming groups with similar features (in our case the trajectory) between each others instead of within the group itself.

4 | Least-Square optimisation for parameters extraction from experimental data

The main goal of this section was to extract the initial ball height h_0 as well as the coefficient of restitution ρ from the experimental data resulting from previous sections 2 and 3. This has been achieved by trying to fit as best as possible the experimental data with a theoretical formulation of the problem. A least-square optimisation has been chosen for this purpose and implemented in MATLAB with the optimisation toolbox *MPT*.

4.1 Theoretical formulation

In order to deduce parameters from the experimental data, a theoretical formulation of the problem was first elaborated.

The computational complexity of the problem being commonly greatly increased when introducing a discrete variable, it has been chosen to solve the problem locally. The data on which the optimisation is applied has therefore been restricted to the time of one bounce. This way, the number of the current bounce n could be kept constant and a discrete variable was not requested anymore. The computational efforts are thought to have been greatly reduced by this choice. Inconsistencies could however occur between each bounce as the parameters evaluated on solely one bounce could have been highly different if evaluated on other bounces. This impact has however been deemed negligible as the data-set is thought to be most reliable for the 2 first bounces of the ball, which lead to globally similar results as proven in section 4.3.

Considering the initial velocity at the beginning of the n^{th} bounce v_n and the gravity g, the height of the ball through time $h(t)$ could be deduced. The time of the start of the bounce is considered to be at 0 s in the following expression:

$$h(t) = v_n t - \frac{gt^2}{2} \tag{4.1}$$

In order to evaluate v_n, an energetic equilibrium has been used. Initially, only potential energy E_p is present. It is given as follow in function of the mass of the ball m and its initial height with respect to the ground h_0:

$$E_p = mgh_0 \tag{4.2}$$

Not considering losses in a first time, the potential energy is transformed entirely in kinematic energy E_c once the ball touches the ground:

$$E_c = \frac{mv_n^2}{2} \tag{4.3}$$

By equaling the two expressions, the velocity at the beginning of the n^{th} bounce without considering losses v'_n can be estimated as follows:

$$\frac{mv'^2_n}{2} = mgh_0 \rightarrow v'_n = \sqrt{2gh_0} \tag{4.4}$$

By adding the coefficient of restitution ρ to represent the loss of energy each time the ball touches the ground, the following equation can be found:

$$v_n = \rho^n \sqrt{2h_0 g} \tag{4.5}$$

As for the initial height h_0, due to the experimental nature of the data-sets, its units (in pixels) is not really meaningful. This height should therefore be converted in a more conventional unit system : in meters. Thanks to the knowledge of the actual gravity on Earth ($g_{real} = 9.81 m.s^{-2}$) and the gravity resulting from the optimisation g in $pixels.s^{-2}$, the conversion between meters and pixels can be determined. The initial height h_0 can therefore be converted in meters h_{real} as followed:

$$g_{real} = 9.81 = \alpha g \rightarrow h_{real} = \alpha h_0 = \frac{9.81}{g} h_0 \tag{4.6}$$

4.2 MATLAB implementation

Following the equation 4.1, the error between the theoretical and experimental data for the height ξ_i can be estimated for each sample i of the data-set. The Least-Mean square of this error has been chosen as the cost function for our optimisation problem. The parameters to optimize are therefore the initial velocity at the beginning of the n^{th} bounce v_n and the gravity g. Those parameters are constrained to be above or equal to 0. We don't expect those parameters to be equal to 0 but a non-strict equality is necessary in order to get a convex optimisation. The optimisation is then indeed convex as h_{th} is affine with respect to the parameters v_n and g. This formulation of the problem in a convex form greatly facilitates computation as convex optimisation is known to be easily solved. The formulation of the optimisation problem is summarized in Figure 4.1. A MATLAB code based on this formulation has then been implemented using the MPT toolbox. This code can be found in the provided GitHub.

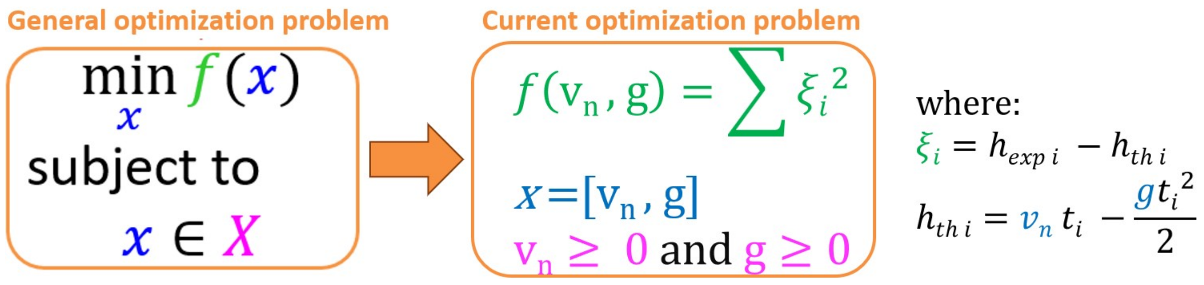
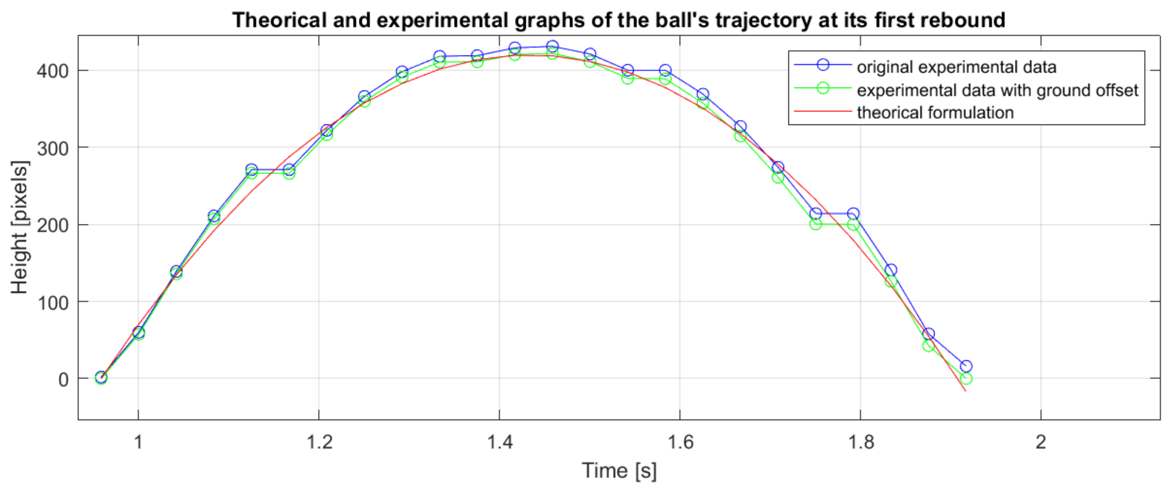

General optimization problem
$\min_x f(x)$
subject to
$x \in X$
Current optimization problem
$f(v_n, g) = \sum \xi_i^2$
$x = [v_n, g]$
$v_n \geq 0$ and $g \geq 0$
where:
$\xi_i = h_{exp\,i} - h_{th\,i}$
$h_{th\,i} = v_n t_i - \frac{g t_i^2}{2}$

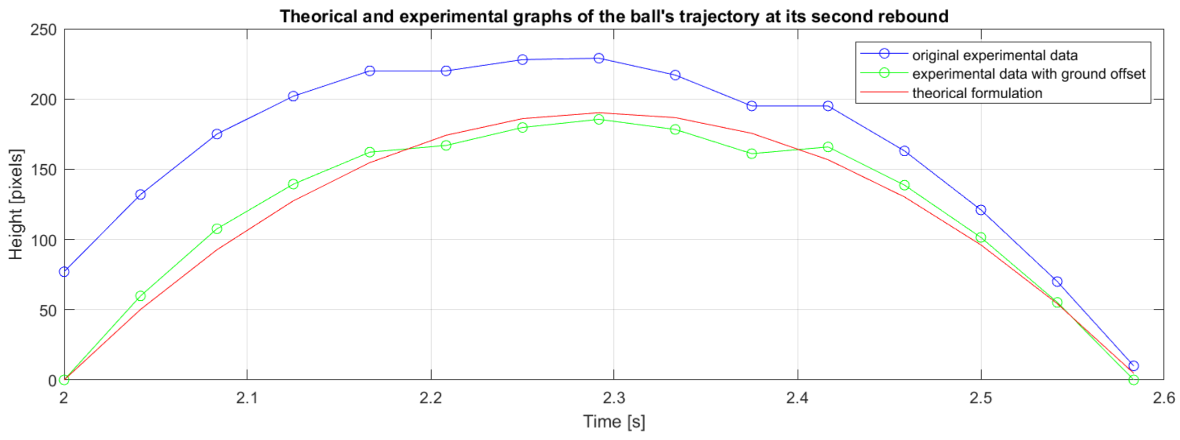
Figure 4.1: Scheme of the formulation of the optimisation problem.

4.3 Discussion of the results

The optimisation was applied to the first and second bounces of the data-set *ball_1_3*. The data had to first be treated in order to remove the potential outliers and the ground offset (cf. section 3). Afterwards, the optimal values for the parameters v_n and g were found thanks to the MATLAB code implemented. The Figure 4.2 shows a comparison between the experimental trajectory of the ball tracked with a camera (before and after treating it) and the reconstructed curved based on the "optimised" parameters found based on the theoretical formula (cf. Equation 4.1). It can be observed that the offset of the ground has more impact in the second bounce. This is due to the fact that the ball has moved closer to the camera during the second bounce, while during the first bounce it remained approximately at the same distance. The theoretical curves (in red) are also shown to follow rather well the experimental curve on which the optimisation was performed (in green). This validates the theoretical formulation of the problem as well as the optimisation methodology.



(a) First bounce.



(b) Second bounce.

Figure 4.2: Graphs of the experimental and reconstructed theoretical trajectories for the data-set *ball_1_3* during its first and second bounces.

Once the methodology was validated through an observation of the graphs, the actual parameters of interest (ρ and h_real) could be extracted from the parameters resulting from the optimisation (v_n and g). The several results obtained are listed in Table 4.1.

Bounce n	**Gravity** g $[pixels.s^{-2}]$	**Initial height** h_0 $[pixels]$	**Initial height** h_{real} $[m]$	**Bounce velocity** v_n $[pixels.s^{-1}]$	**coefficient of restitution** ρ [-]
1	3 731.4	702	**1.8456**	1 770.1	**0.7733**
2	4 415.2	702	**1.5598**	1 296.0	**0.7215**

Table 4.1: Resulting parameters evaluated after Least-Square optimisation of the first and second bounces for the data-set *ball_1_3*.

The coefficient of restitution ρ was first extracted from v_n using the Equation 4.5 previously established. The tennis ball colliding with the wooden floor was estimated to have a coefficient of restitution ρ of 0.7733 and 0.7215 based on the first and second bounces respectively. The actual coefficient of restitution should be around 0.75 [1]. The results found for the coefficient of restitution are therefore rather coherent.

[1] `https://physics.stackexchange.com/questions/256468/model-formula-for-bouncing-ball?fbclid=IwAR2XVqx1LKmuw6xDIkLWjOo10P-h50s_c6igJVj1wYTzfNPkiAIeF4bOUqQ`

As for the initial height of the ball –at which the ball was released–, the results were expected to be between 1.4 and 1.5 meters. It has however been evaluated to be 1.8456 and 1.5598 meters according to the first and second bounces data-sets. While the second bounce results are rather coherent, the resulting height from the first data-set in thought to be the result of the ball slightly touching the wall at its first bounce. This would have in term slowed the ball and therefore increased the time of falling, in term decreasing the apparent gravity. This could also be the result of not enough data points taken to have a reliable enough time scale.

5 | Parameter estimation by application of LPCA method on the ball velocity

In Chapter 4, the parameters of the system have been determined by solving a least-square optimisation problem based on the ball vertical position. In this chapter, an alternative method is proposed to estimate two parameters: the gravity g and the coefficient of restitution of the ball ρ. The proposed method is to apply Local PCA to approximate the velocity of the ball and from this approximation to deduce the parameters.

5.1 Outlier detection and cleaning

First, the velocity of the ball has been computed from its positions. As shown in Figure 5.1, some outliers can be observed. These outliers are detected and cleaned using a self-written MATLAB function. The idea behind this outlier cleaning function is that if the point is too far from its two (left and right) neighbours, it is considered an outlier.

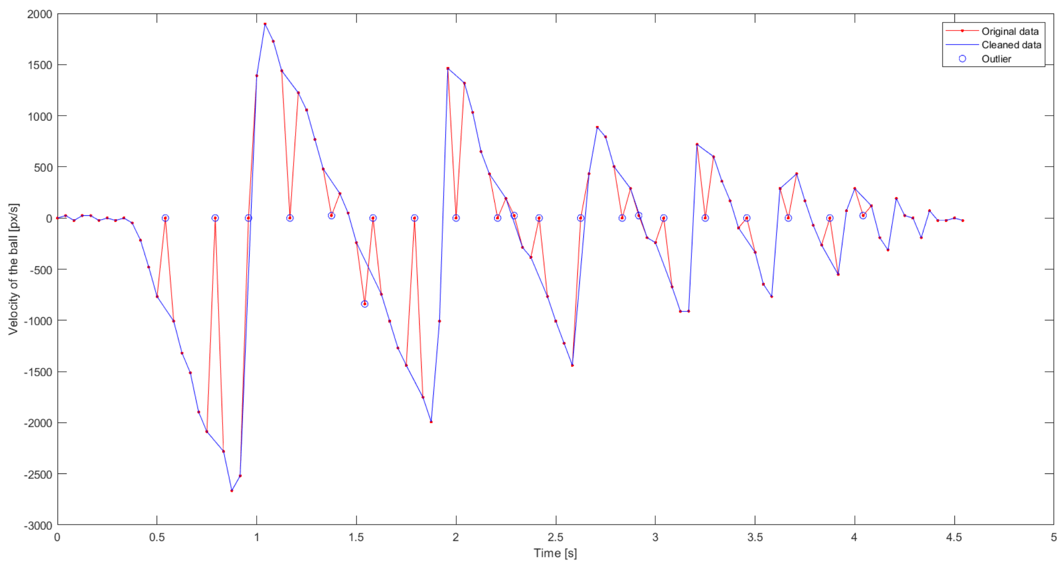


Figure 5.1: Results of the self-written outlier cleaning function.

5.2 Clustering

Next, some clusters are built using DBSCAN clustering method. This method has been explained previously in Section 3.2. From this clustering, it results 5 clusters shown in Figure 5.2. The points in blue correspond to noisy samples that have not been assigned to a cluster by the DBSCAN algorithm.

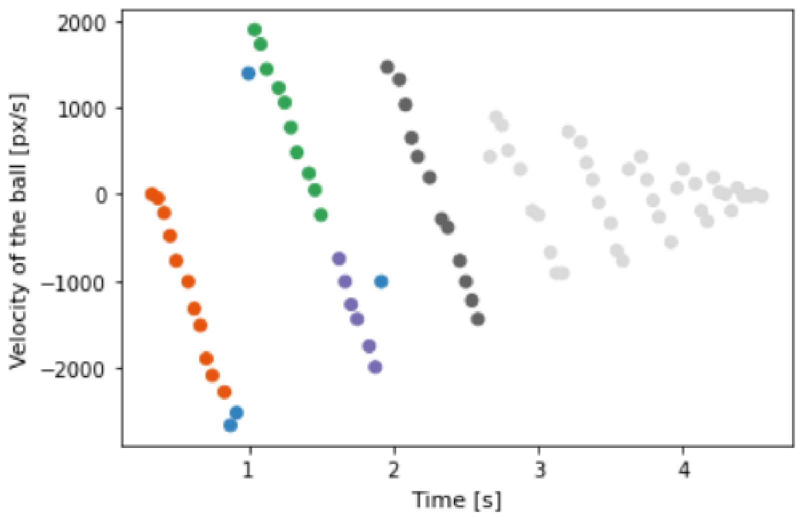


Figure 5.2: Clustering of the ball velocity dataset using DBSCAN.

5.3 Local PCA & Parameter estimation

Finally, the PCA algorithm can be applied in each cluster. The results are shown in Figure 5.3.

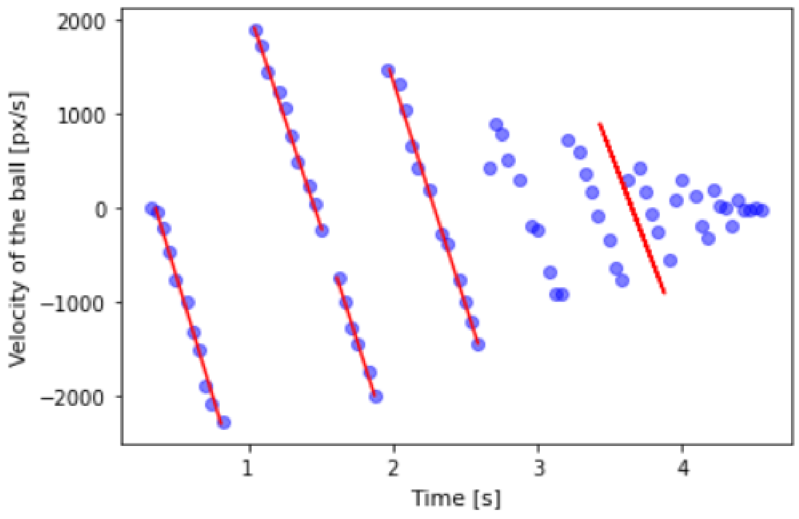


Figure 5.3: Results of the application of Local PCA on the velocity dataset.

From the Local PCA results, the parameters of the system can be estimated. Since the last cluster is poorly approximated by Local PCA, only the four first clusters are considered to estimate the parameters.

The standard gravity g can be estimated knowing that $-g$ is the slope of the curve of the velocity v with respect to the time t (since $v(t) = v_0 - gt$). The coefficient of restitution ρ is the ratio between the velocity of the ball just after the bounce and the velocity of the ball just before the bounce. The following results are obtained: $g = -4714.69$ px/s^2 and $\rho = 0.8043$.

To validate the results and verify if they correspond to reality, the following method is proposed. In the video, the units used for length and time are pixels (px) and seconds (s), respectively. To be able to compare the length values of the video with reality, it is necessary to be able to convert pixels into meters. To do this, since we know the gravity factor in our real world and since the time units are the same in the video and in reality, the lengths can be converted as follows: $L_{\text{meters}} = \frac{9.81}{g} L_{\text{pixels}}$, with g the standard gravity (in px) estimated earlier.

The heights obtained are $h_0 = 1.147$ m for the initial height, $h_1 = 0.876$ m for the first bounce and $h_2 = 0.473$ m for the second bounce. These heights seem coherent with reality if we roughly estimate the actual heights using the meter visible in the video. As for the value of the coefficient of restitution, a value of $\rho = 0.75$ for a tennis ball has been found on the Internet, which is quite close to the result obtained here.

6 Regression for ball trajectory prediction

Several regression methods were used in order to deduce the future vertical coordinate of the ball h_{i+1} based on its current position h_i and 2 previous positions h_{i-1} and h_{i-2}. It has been implemented in a Python code where several *sklearn* modules were used and applied on the data-set *ball_1_3*.

The data was first treated in order to form, from the height measurements, a matrix with 3 features: h_{i-2}, h_{i-1} and h_i. The matrix of labels consisted then in the future positions h_{i+1}. The data was then split in two: 80% in a set for training and 20% for testing.

6.1 Least-Square regression

The main idea behind linear regression is to find a simple and adequate representation of the effect of the inputs x_i on the outputs y_i. It has the advantage of having low bias since the error is minimized with Least-Mean-Square. A large variance –meaning large uncertainties– is however to be expected due to the fact that variables interact with each other. The linear regression works in general the best on the data it trains on but is usually not so good when it comes to new data which is rather inconvenient in our case as the goal is to predict the trajectory. Another drawback of least-square estimation is the difficulty to interpret the result in case of a large number of predictors.

A linear regression assumes that the regression function is linear with respect to the inputs in order to deduce the adequate parameters β_i.

$$f(x) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j \tag{6.1}$$

After a scaling of the data with the sklearn's module *StandardScaler()*, a least-square regression was performed. The Least-Square regression tries to minimise the function $RSS(\beta)$ defined as:

$$RSS(\beta) = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{6.2}$$

Although the least-square estimation has low bias, it can be observed that it presents a large variance in Figure 6.1. The line in orange represents the linear regression obtained from the 80% reserved for training while the blue dots represent the 20% of the data that have been kept for testing.

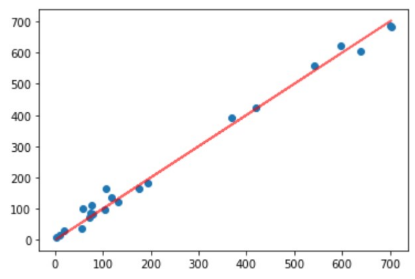


Figure 6.1: Graph of the Least-Square regression.

6.2 Lasso and Ridge regression

Contrary to the least-square estimation, small details can indeed be neglected in the data in favour of recovering the overall picture. Prediction accuracy can therefore be improved by shrinking or even setting some coefficients to zero. Some bias is introduced in favour of reducing the variance of the predicted values. The idea behind Lasso and Ridge regressions is therefore to decrease the variance by shrinking the data.

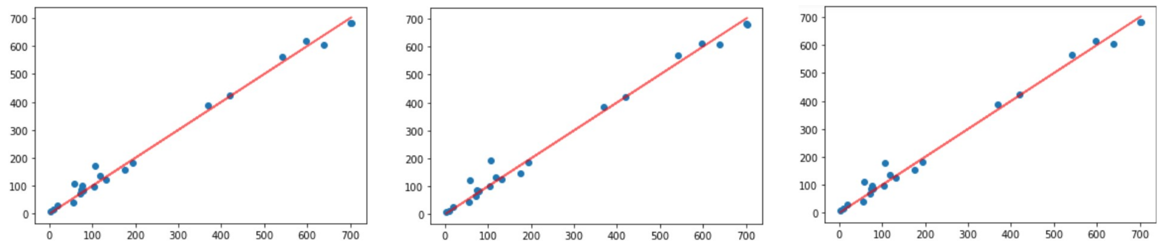
$$RSS = (y - X\beta)'(y - X\beta) + \lambda\beta'\beta \rightarrow \hat{\beta} = (X'X + \lambda I)^{-1}X'y \tag{6.3}$$

$$X = WLA' \rightarrow X\hat{\beta} = WW'y \tag{6.4}$$

The data can be shrunk by adding a penalty term λ that aims to minimize the data corresponding to variables with a smaller covariance l_i. While Lasso minimizes the sum of coefficient by an iterative process, the Ridge minimize the sum of coefficient squared. Contrary to Ridge, some coefficients in Lasso can become totally 0. Ridge therefore applies a continuous shrinkage contrary to Lasso.

$$X\hat{\beta}_{bridge} = \sum_{j=1}^{p} w_j \frac{l_j^2}{l_j^2 + \lambda} w_j' y \tag{6.5}$$

$$\hat{\beta}_{lasso} = min_{\beta}(\frac{1}{2}\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}|\beta_j|) \tag{6.6}$$



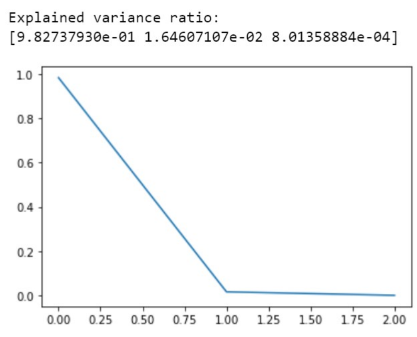
(a) Lasso. (b) Lasso with cross-validation. (c) Ridge with cross-validation.

Figure 6.2: Graph of the Lasso and Ridge regressions with and without cross-validation.

Figure 6.2 shows the different results for different regressions. All results are quite similar and almost identical for Ridge and Lasso with cross-validation.

6.3 Principal component regression

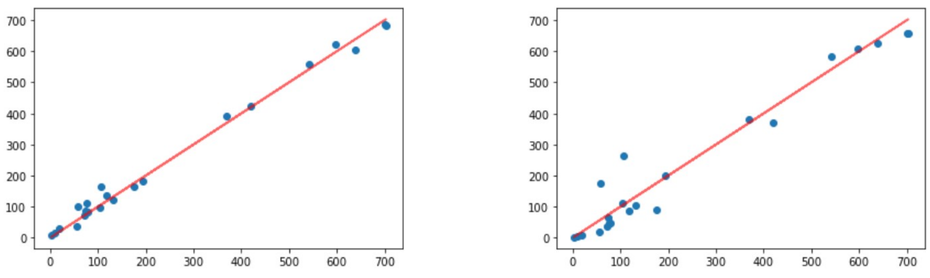
While Ridge regression shrinks the regression coefficients of the principal components, Principal component regression truncates them. The advantage of this method is that features become uncorrelated and dimensionnality can be reduced.



```
Explained variance ratio:
[9.82737930e-01 1.64607107e-02 8.01358884e-04]
```

Figure 6.3: Graph of percentage of representation of the 3 principal components.

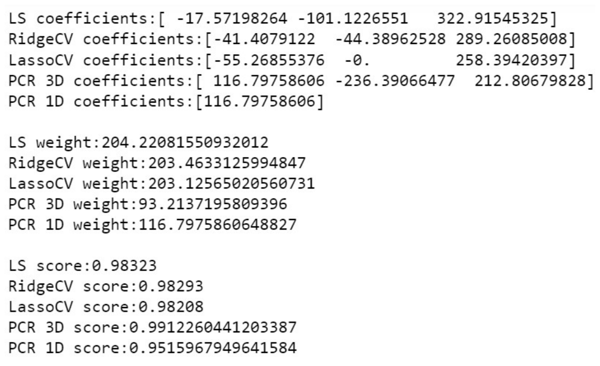
As shown in Figure 6.3, the first component represents 98.3% of the data. The two other components can therefore be removed in order to reduce the dimension to 1. By cancelling some coefficients, the regression is more abrupt and lead to large variances in the prediction because variables are either retained or discarded. The dimension can be reduced before the regression as shown in Figure 6.4. By keeping only one principal component out of the three, the data is more sparse. The resulting linear regression is therefore less good.



(a) Without dimension reduction. (b) With dimension reduction to 1.

Figure 6.4: Graph of the principal component regression before and after reduction of the dimension.

6.4 Discussion of the results



```
LS coefficients:[ -17.57198264 -101.1226551   322.91545325]
RidgeCV coefficients:[-41.4079122  -44.38962528 289.26085008]
LassoCV coefficients:[-55.26855376  -0.         258.39420397]
PCR 3D coefficients:[ 116.79758606 -236.39066477  212.80679828]
PCR 1D coefficients:[116.79758606]

LS weight:204.22081550932012
RidgeCV weight:203.4633125994847
LassoCV weight:203.12565020560731
PCR 3D weight:93.2137195809396
PCR 1D weight:116.7975860648827

LS score:0.98323
RidgeCV score:0.98293
LassoCV score:0.98208
PCR 3D score:0.9912260441203387
PCR 1D score:0.9515967949641584
```

Table 6.1: Coefficients, scores and weights of the various regression methods.

In Table 6.1, it can be observed than Lasso has a worst score than Least-Square and Ridge. Nevertheless, Lasso is still interesting to use has it is a simpler model since it use less features (since one of its coefficient is 0, leaving only 2 non-zero features). Lasso is therefore better than Least-Square in avoiding overfitting.

As for the Principal component regression, the reduction of dimension influences the result by reducing the score from 0.99 to 0.95. However this impact is rather small and a rather good regression is still possible with only this first principal component.

7 | Ball characteristics extraction with PCA

In this chapter, the aim is to simulate a model of the bouncing ball and make a data set according to different parameters and ball characteristics. Simulation is required since creating a large and diverse enough data set from a experimentation is highly time consuming and difficult.

7.1 Simulation

The physic of bouncing ball is a remarkable topic to study since it demonstrates several interesting dynamical concepts relating to acceleration, momentum, and energy. Moreover, a bouncing ball model, containing continues dynamic and discrete transition, is a simple and classical example of hybrid system. In this section, the dynamic model of the bouncing ball without considering the air drag is performed. Neglecting the air drag leads to consider the ball as a particle and indeed the mass and its corresponding inertia do not play a role in the dynamic behaviour of the system. Also, it is assumed that the ground is perfectly rigid.

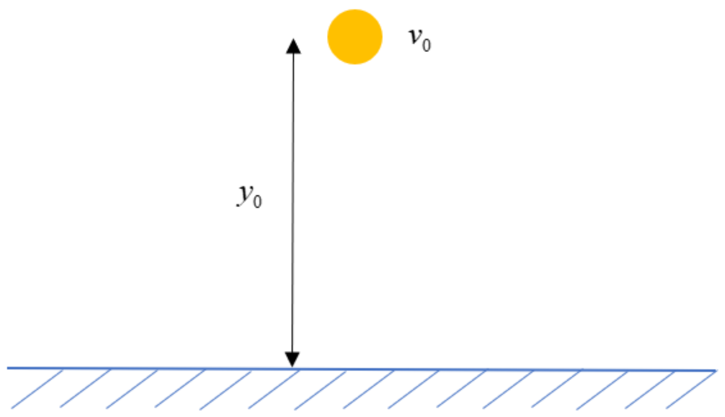


Figure 7.1: A ball is projected with initial height and velocity of y_0 and v_0, respectively.

The objective of simulating the bouncing ball is to create a data set based on different initial conditions such as velocity and height. To make the simulation more realistic, the time when the ball hits the ground and dissipates energy, which is considered as a function of its velocity and coefficient of restitution, has been added to the total simulation of time. The balls are also subjected to different values of gravity as if they were simulated in different planets. Furthermore, different balls in the simulation are categorized based on the coefficient of restitution of the ball. Considering the initial velocity, the velocity at each bounce obtained in equation 4.5 becomes:

$$v_n = \rho^n \sqrt{2h_0 g + v_0^2} \tag{7.1}$$

The height of each bounces can be found as follows:

$$h_n = \frac{(\rho^n v_1)^2}{2g} \tag{7.2}$$

The time when the ball hits the ground and reaches the position where its velocity is zero can be calculated as follows:

$$t_n = \frac{\rho^n v_1}{2g} + \Delta t'_n \tag{7.3}$$

where $\Delta t'_n$ is the collision time. One can also find the analytical solution for the time at which the ball settles down. This time is the sum of an infinite geometric sequence which is given by:

$$T = \frac{1}{g}\left[v_0 + v_1 \frac{1+\rho}{1-\rho}\right] + \sum_{n=1}^{\infty} \Delta t'_n \tag{7.4}$$

Equation 7.4 is added with summation of all collisions during the simulation. The whole time of the simulation can now be considered as one specific feature of a released ball. In addition, each height of bounce calculated in equation 7.3 can be considered as a feature. Another interesting feature that can be extracted from the simulation is the number of bounces during each trial, which can be obtained as follows:

$$n = \frac{\ln \frac{2gh'}{2gh_0 + v_0^2}}{2\ln\rho} \tag{7.5}$$

where h' is the lower limit for the height of the bouncing ball in the experiment.

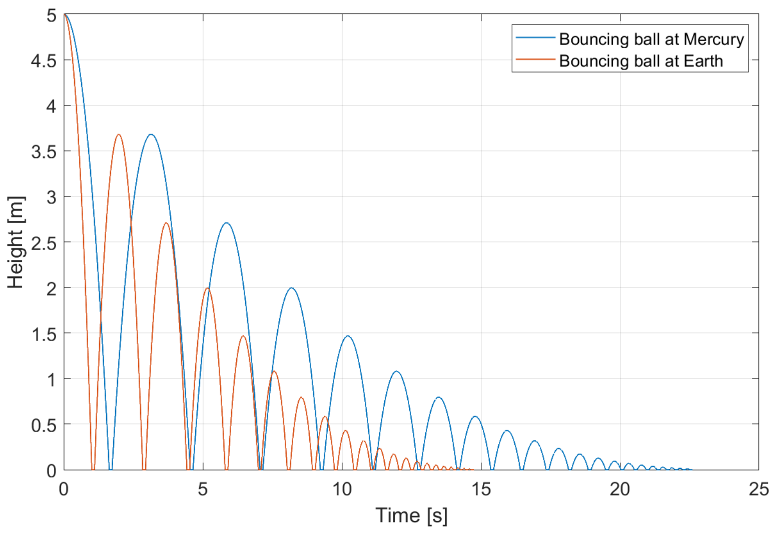
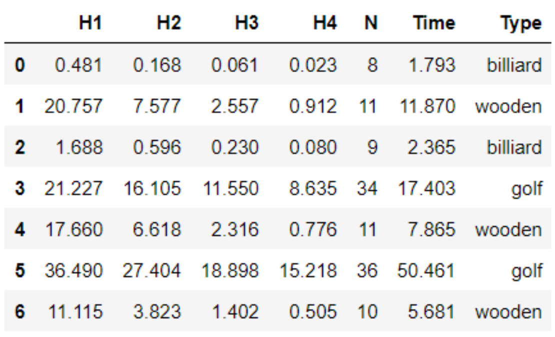


Figure 7.2: Similar simulations of the bouncing of a golf ball in different situations.

7.2 Principle Component Analysis - PCA

Principle Component Analysis (PCA) is used in exploratory data analysis and for making predictive models. It is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. The first principal component can equivalently be defined as a direction that maximizes the variance of the projected data. The i-th principal component can be taken as a direction orthogonal to the first i-1 principal components that maximizes the variance of the projected data.

PCA was use here to reduce the data dimension. Six features are presented in the generated data set and shown in figure 7.3. They correspond to the first 4 bouncing heights (H_1, H_2 H_3 and H_4), the last bounce until the limit is reached N, and the total corresponding consumed time *Time*.



	H1	H2	H3	H4	N	Time	Type
0	0.481	0.168	0.061	0.023	8	1.793	billiard
1	20.757	7.577	2.557	0.912	11	11.870	wooden
2	1.688	0.596	0.230	0.080	9	2.365	billiard
3	21.227	16.105	11.550	8.635	34	17.403	golf
4	17.660	6.618	2.316	0.776	11	7.865	wooden
5	36.490	27.404	18.898	15.218	36	50.461	golf
6	11.115	3.823	1.402	0.505	10	5.681	wooden

Figure 7.3: Dataset.

The mathematical formula for PCA is:

$$\mathbf{Z} = \mathbf{X} \cdot \mathbf{A} \tag{7.6}$$

In which:

- X is the matrix of the data set.
- a_n is chosen to maximise $var(\mathbf{z_n}) = \bar{\mathbf{z_1^n}} - \bar{\mathbf{z_n}}^2 = \mathbf{a_n^T \Sigma a_n}$, subject to $\mathbf{a_1^T} \cdot \mathbf{a_n} = \mathbf{1}$ ($\mathbf{\Sigma}$ refers to the covariance matrix of the data set).

The PCs selection is based on the cumulative percentage of the total variance. Formula 7.7 gives the relation between the eigenvalues of the covariance matrix and the variance of the original data set **X**, and 7.8 shows the desired fraction of the total variance. Figure 7.4 was generated based on the calculation.

$$\Sigma_{k=1}^{p} l_k = \Sigma_{j=1}^{p} var(x_j) \tag{7.7}$$

$$t_q = \frac{\Sigma_{k=1}^{q} l_k}{\Sigma_{k=1}^{p} l_k} \tag{7.8}$$

From the figure 7.4, one can find that with the first 2 principle components, around 93% of variance can be described. It can also be noticed than 3 of them represent by themselves around 96% of the data. Since, the desired variance is 95% in our case, only 3 PCs were kept to generate the matrix **Z**.

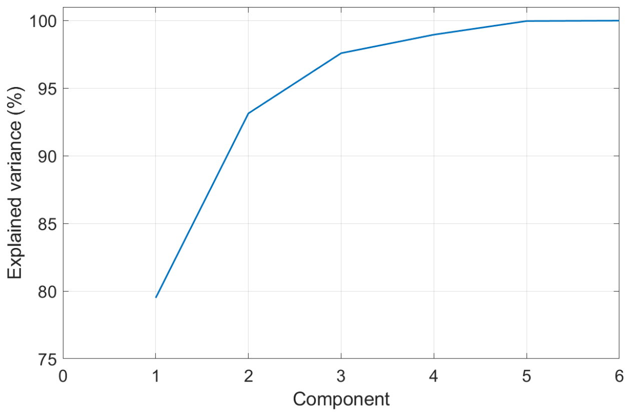


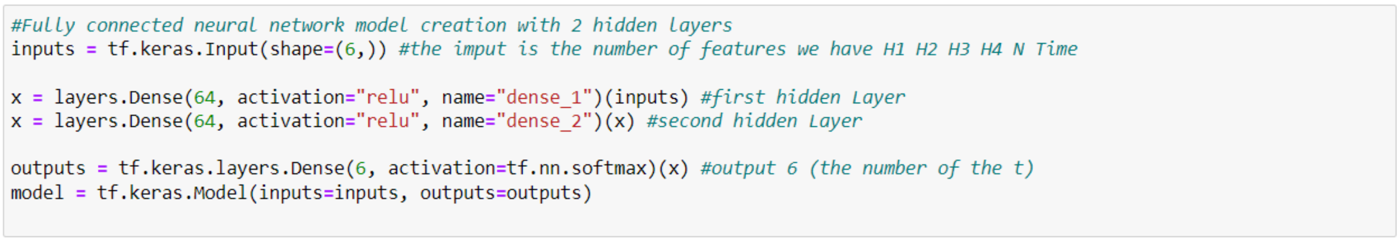
Figure 7.4: Variance in percentage.

8 Ball characteristics prediction with neural network

In this section, the aim is to predict the type of the ball according to the features extracted from the simulation. To do so, a multi-layer fully connected neural network was implemented, trained, and validated.

8.1 Creation of the model

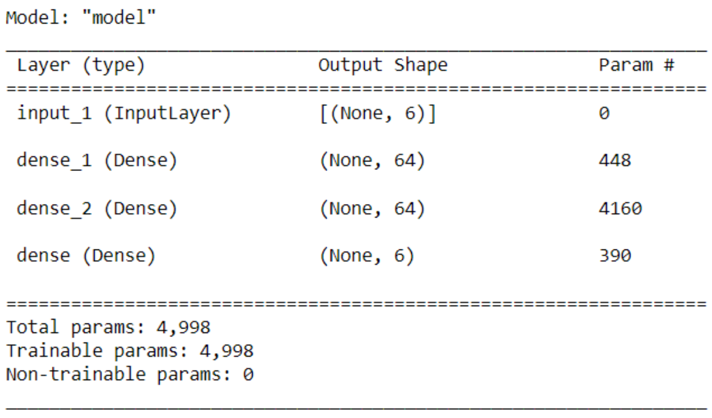
To build the model, Karas function –a library avalable in the tensorflow– is used. Two hidden layers using "relu" activation function are defined with six inputs representing the features and six outputs corresponding basically to the type of the ball. Technically, there is no specific rule to define how many hidden layers a model needs; however, the probability of over fitting will be increased by adding more hidden layers on the one hand. Having less hidden layers may reduce the accuracy of the model on the other hand.



```
#Fully connected neural network model creation with 2 hidden layers
inputs = tf.keras.Input(shape=(6,)) #the imput is the number of features we have H1 H2 H3 H4 N Time

x = layers.Dense(64, activation="relu", name="dense_1")(inputs) #first hidden Layer
x = layers.Dense(64, activation="relu", name="dense_2")(x) #second hidden Layer

outputs = tf.keras.layers.Dense(6, activation=tf.nn.softmax)(x) #output 6 (the number of the t)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

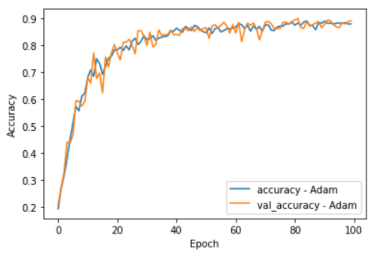
Figure 8.1: Creating the models - hidden layers, inputs and outputs of the neural network.

Table 8.1 shows the summary of the model–containing 6 input layers, two hidden layers which contains 64 neurons dense layer for each, and 7 neurons dens layer for the outputs–which contains 4998 yet-to-be-trained parameters. In the next section, these parameters will be trained by the training test data set.

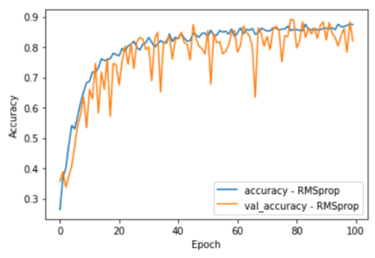


```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 6)]               0

 dense_1 (Dense)             (None, 64)                448

 dense_2 (Dense)             (None, 64)                4160

 dense (Dense)               (None, 6)                 390

=================================================================
Total params: 4,998
Trainable params: 4,998
Non-trainable params: 0
_________________________________________________________________
```

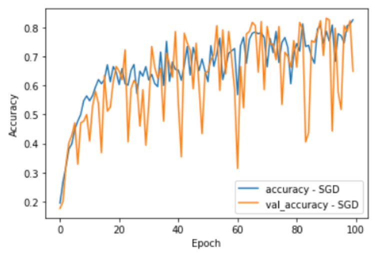
Table 8.1: Model - Input, hidden layers, output ,and parameters.



(a) Adam optimizer

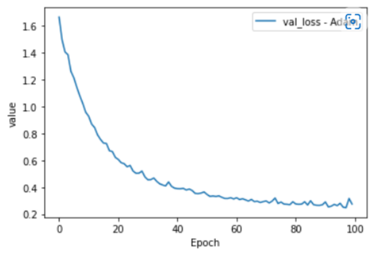


(b) RMSSprop optimizer

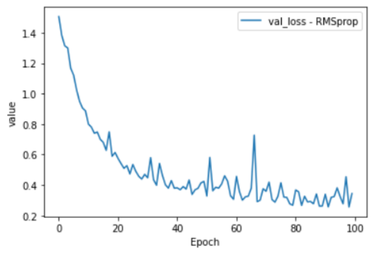


(c) SGD optimizer

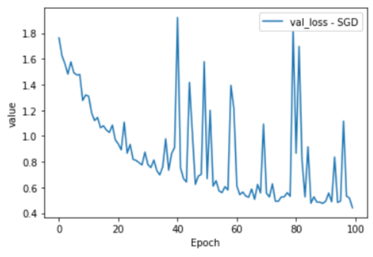
Figure 8.2: Comparison Between different optimizers.



(a) Adam optimizer



(b) RMSSprop optimizer



(c) SGD optimizer

Figure 8.3: Comparison between different losses.

8.2 Training the model

To train the neural network we have to define a function that has the quality of the neural network which shows that the model is predicting correctly. Categorical-cross entropy, a loss function that is used in multi-class classification tasks, shows how good the model is trained. It is defined as follows:

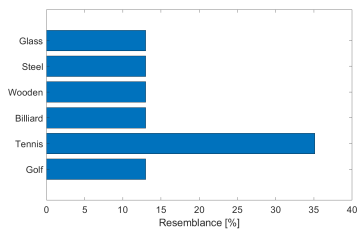
$$Loss = -\sum_{n=1}^{n} y_i \log y_i' \tag{8.1}$$

In this formulation, y_i is the predicted value and y_i' is a correct observation. The objective is always to minimize the loss function as much as possible. Having selected the loss function, we need to assign a optimizer. The optimizer is algorithms to reduced the loss which was already obtained by Categorical-cross entropy loss function. There are quite a lot of optimizers available in the tensor flow such as Adadelta, Adagrad, Adam and SGD. In this project, the model is trained with different optimizers to observe which one is the best fit to the created model. There are 1350 observations in the data set. 1000 of them (74%) are used to train the network as the remaining part (26%) are used to validate the system.

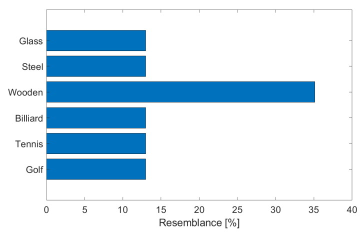
Investigation illustrates that Adam optimizer, involving a combination of two gradient descent methodologies, shows a better performance in comparison between the other two methods, SGD and RMSSprop methods. Figure8.3 shows the loss function optimized by the Adam optimizer. The loss function as expected is decaying and converging after almost 80 epochs. Moreover, the accuracy and validation accuracy are converged almost in the first 60 epochs, and to avoid overfitting and reducing computational cost and time 80 epochs are enough. Also, to avoid overfitting, weight decay factor can be applied to convolutional layers. As the figure 8.2 shows, with the increase of epoch, the validation accuracy converge to around 0.87, which is the same as the accuracy driven from the training data set.

8.3 Validation of the model

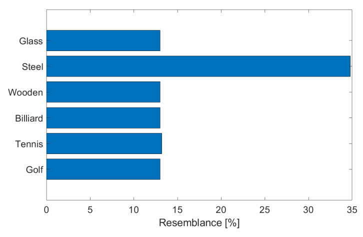
To validate the model the 6 features of the different balls were put as inputs to the model. Figure 8.4 shows the presentation of resemblance of each ball with the predicted model. Predictions by the



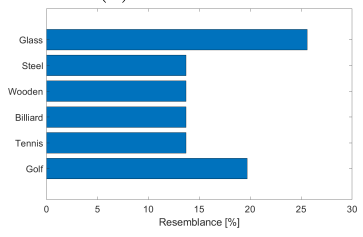
(a) Tennis ball



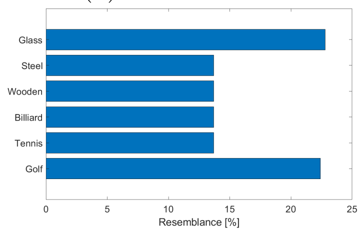
(b) Wooden ball



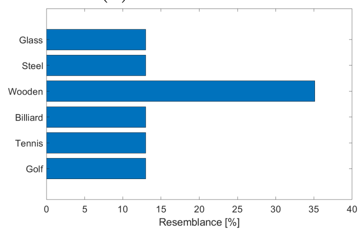
(c) Steel ball



(d) Glass ball



(e) Golf ball



(f) Wooden ball

Figure 8.4: Validation of the model.

neural network shows that it has trained sufficiently. There is only one prediction missed between the glass ball and golf ball. Other predictions are in good agreement with what was expected.

9 | Conclusion

To conclude, all the objectives were fulfilled except for the clustering of the ball bounces which is not optimal. Another splitting method than clustering should be indeed used as the features of each bounces are too similar to be distinguished effectively based on it. Thanks to an efficient tracking of the ball based on its color, experimental data could be acquired and treated automatically which is essential for data-driven as a lot of data is required and a fast process of acquiring and treating the data is therefore essential. the treatment of the experimental data is also not to be taken slightly since non-visible outliers should be absolutely disregarded. Due to the choice to use computer vision, other problems of interpretation could also occur as the distance to the camera might lead to deceiving result. Indeed, the ball can move slightly towards the camera during the experiment. A different floor reference should therefore be used for each bounces in order to extract efficiently the actual height of the ball. In the end, with only a video of the ball bouncing on the floor, its future trajectory can be estimated with a linear regression. Several parameters such as the initial height from which the ball was released (without any knowledge of the distance from the camera,nor any reference for the length in the video). This is possible through the knowledge of the physics behind it and a Least-square optimisation. The coefficient of restitution –corresponding to the quantity of energy lost by the ball at each contact with the floor– can be determined in a similar way.

However, the knowledge that can be extracted from computer vision are also limited. Simulations of the ball were therefore also performed in order to evaluate additional characteristics. First, an analytical solution of a bouncing ball considering some nonlinearities derived. Then, according to the characteristic of the different balls, the data set based on different initial conditions has been created. Neural network based on the known data set has been created, trained, and validated. To optimize the loss function, different methods has been used and compared. Among the selected well known optimizer, Adam has shown the better performance. The trained model has predicted the characteristic of the ball based on the features of the ball very well. Also, to reduced the dimension of the data set PCA has been applied to reduce the computation cost and time for training the neural network; however, the new data set was not able to trained the model well, and consequently the prediction was not successful.