TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

AI in Digital Car Design for Pedestrian Safety

Authors	Chen Xu, Katharina Oberst, Shubham Khatri, Willem van Hove
Mentor(s)	PhD. Candidate Laure Vuaille (BMW Group)
Co-Mentor	Dr. Nada Sissouno (Department of Mathematics)
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Feb 2020

Abstract

This project investigates the use of data-driven digital car design for pedestrian safety. The BMW group provided the data for the project from finite element simulations. The goal of this project is that given an unknown car, the most similar car can be fetched from the data lake. Next, the differences between these two cars can be compared. We tackled this problem by developing an algorithm that finds the closest car to a provided unknown car and then computes global and localized geometric as well as the non-geometric differences between the two. These results are provided in a concise human-readable format.

We perform several data preprocessing steps, which are crucial for adequate change detection. In order to find the closest matching car, we extract high level geometric features from the point cloud. We employ a mixture of the modified K-Nearest Neighbor and the Random Forest Model to find the closest matching car using the extracted geometric features. Using the detected closest car and the unknown car, we extract the global differences. To extract the localized differences, we perform region splitting and compute the differences in the extracted regions. We use a combination of slicing and Quadtree to perform region splitting. In each of these regions, we use Hausdorff distance to detect the geometric differences and K-Dimensional Tree along with Sparse Change Detection to find non-geometric changes. The pipelined model was tested on a test set of 10 new point clouds, where we verified the accuracy of the developed algorithm. We were able to verify that the developed algorithm can detect major as well as minor changes efficiently and accurately and present them in a human readable manner.

Keywords: Point Cloud Comparison, Point Cloud Registration, Point Matching, Slicing, Correspondence.

Contents

1   Introduction

About 14% of all road accidents involve pedestrians and happen in urban areas where speeds are moderate. The head counts as one of the most frequently injured body region. Therefore, strict regulations are enforced to minimize the impact on the head during a collision. Abiding by these regulations has become a big part in the car development process [1].

The potential risk of head injury in the event of a vehicle hitting an adult or a child, is estimated using a series of impact tests, which are taken at moderate speed using an adult or child head form impactor [1]. The injury criterion that is used to measure the body form impactor is the Head Injury Criterion (HIC) [17] [7]. The HIC value is based on the average value of the acceleration over the most critical part of the deceleration. Regulations worldwide prescribe a maximum allowed HIC value.

The design of the anterior of a car is crucial for pedestrian safety during a vehicle-pedestrian accident. In order to reduce the risk of injury, there are many ways to improve the pedestrian protection performance. This can be done by adapting the design of the engine hoods in various ways, or using new materials designed to have desirable properties [13].

A long iterative design process precedes the introduction of a new car. Crash tests are an important aspect in assessing the safety of a car during the design process. Performing crash tests for each iteration of a car is expensive. Numerical simulations offer a cheap alternative and allow an engineer to predict the outcomes of a crash test. One of the most common simulation methods used is the Finite-Element-Method. Running such a simulation can take up to a full working day.

Currently, the development process relies heavily on human expertise. With technological advancements the possibility to automate the digital development process emerges. Artificial Intelligence (AI) could make the development process less reliant on human expertise. The goal of the project is to implement an intelligent system that is able to identify the similarities and differences of car components between car models. This can be used to aid an engineer to improve the safety of a car.

This paper starts by introducing the data set, and several preprocessing methods are discussed. Next, some state-of-the-art approaches regarding point cloud processing are discussed. Chapter 4 describes the methods applied in this project in detail. These methods are combined in a pipeline consisting of two steps: finding the closest match and local change detection. The performance of the pipeline results are presented, validated and discussed in the final two chapters.

2   Data

For the given problem, we were provided with two data sets, the first being the training set, used for the algorithms that were developed. The second data set was provided for validation of the results obtained from the developed algorithms.

In order to tackle the given problem, it was critical to carry out a preliminary data analysis on data set one, which helped us in gaining an insight into the patterns and distribution of the data. Based on the type and distribution of features, appropriate

choices for preprocessing and algorithms were made, which are described in sections 2.2 and 4.

2.1   Data Set

The two provided data sets come from Finite Elements Models (FEM) developed at BMW to test various properties of the car. The first data set was organized by car families, each having several car models, and each of these car models having several point clouds. They were all clustered similarly based on car family, car model and model iteration as shown in Figure 1.
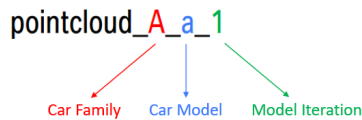

Figure 1: Point cloud labeling.

All the point clouds belonging to a car model were an iteration over the base model, and each of these iterations corresponds to a few geometric changes or property changes.

The first data set consists of a total of 173 point clouds, most of which had approximately $3*10^5$ points. Each point in these point clouds consists of positional information in terms of X, Y, Z coordinates, four bulk properties, and 2 contact artifacts. All the feature names except X, Y, and Z were masked in the provided data.

Since each point cloud consists of a large number of points, each having several features, we decided to analyze the distribution of these features in each point cloud. We started by looking at the distribution of the features in the space. Figure 2 shows the distribution of *var_0* and *var_4* in X and Y directions. We observe that while some of the features are continuously distributed over the space, like *var_0* as shown in Figure 2a and 2c. Others like *var_4* show a discontinuous distribution over the space, indicating sparsity in the feature values.

To further investigate the distribution of the features, we analyze each feature independently using histograms, as shown in Figure 3. We can observe in Figure 3a that the number of points with X close to 1500 is considerably small, which suggests the presence of outliers in the data. We also observe that the *var_0* is highly skewed towards 0, with the existence of some data points having considerably high data values. The data points having higher values either represent outliers in the data set or indicate some component of the car, which has a considerably high value of this variable. We also observe that *var_4* is indeed a sparse feature with 0 values for most of the points. About 1% of the points in a point cloud have a non zero value for *var_4*.

We also decided to employ some preprocessing steps to eliminate outliers, which is further described in section 2.2. In addition to this, wherever necessary, the point clouds were scaled using an appropriate scalar model, to provide equal importance to each feature.

Based on the study and evidence provided above, we classify these variables into two categories, namely *dense* and *sparse*. Sparse features are those who have zero values for a sufficiently large number of points in any point cloud, and dense features are those who have a finite, non-zero value for a sufficiently large number of points in any point cloud.
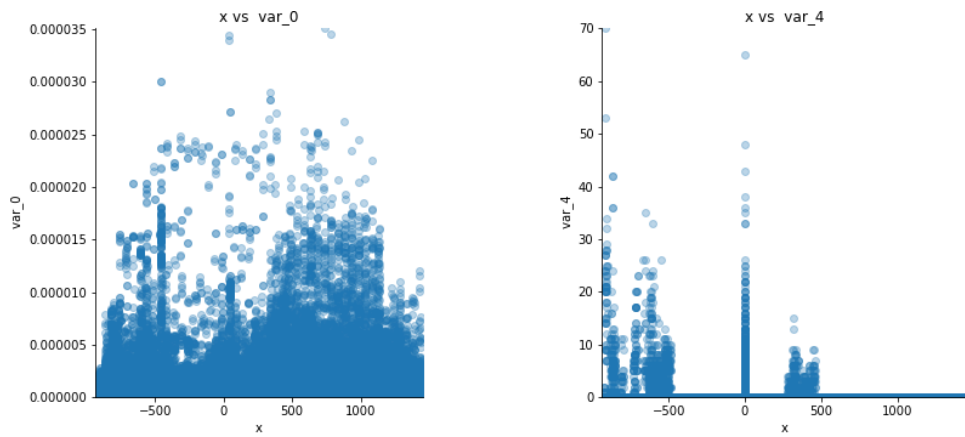
(a) Variation of feature *var_0* across *X*.   (b) Variation of feature *var_4* across *X*.
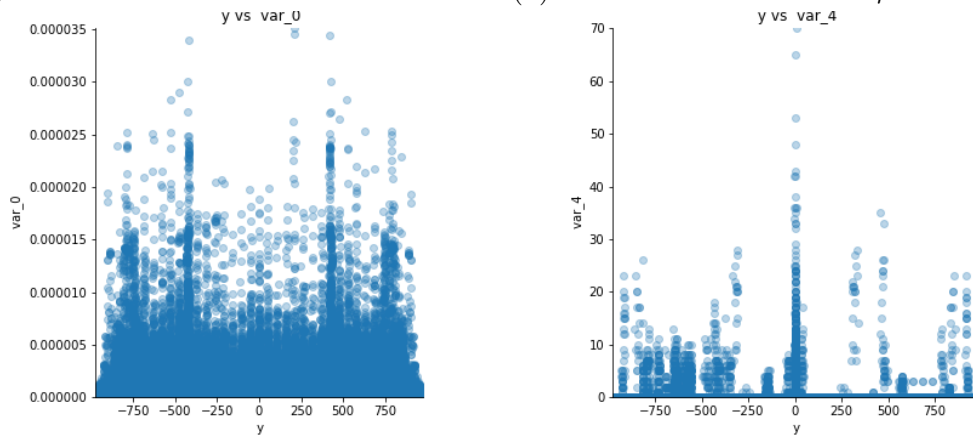
(c) Variation of feature *var_0* across *Y*.   (d) Variation of feature *var_4* across *X*.

Figure 2: Feature distribution plots for *var_0* and *var_4* for all the cars.

The second data set consists of 10 point clouds, the first being the reference, and the other 9 were derived from this car model. Each of these clouds consists of some geometric and non-geometric changes. All the point clouds in this data set have a similar data distribution as the first data set. Additionally, we were provided with a list of changes made to each point cloud and was therefore used for validation of the developed algorithm. In the following sections, these changes are also referred to as labels.

2.2   Data Preprocessing

Based on the preliminary studies described in section 2.1, we carry out a detailed analysis of outlier treatment and faulty data detection. We first look for the faulty point clouds, which could have problems like the absence of windshield or comparatively low point count. This is done to make the model unbiased. Figure 4a shows one such point cloud; we can also see in Figure 4b that some of the provided point clouds, especially car model "A_a", have significantly low relative point count, these point clouds were excluded from the analysis.

After the removal of the faulty point clouds, we perform outlier removal treatment on the

(a) X coordinate distribution.

(b) *var_0* distribution showing skewness towards the 0.

(c) *var_4* distribution showing most of the values are 0.

(d) Zoomed plot for *var_4*, indicating the sparsity in the values.
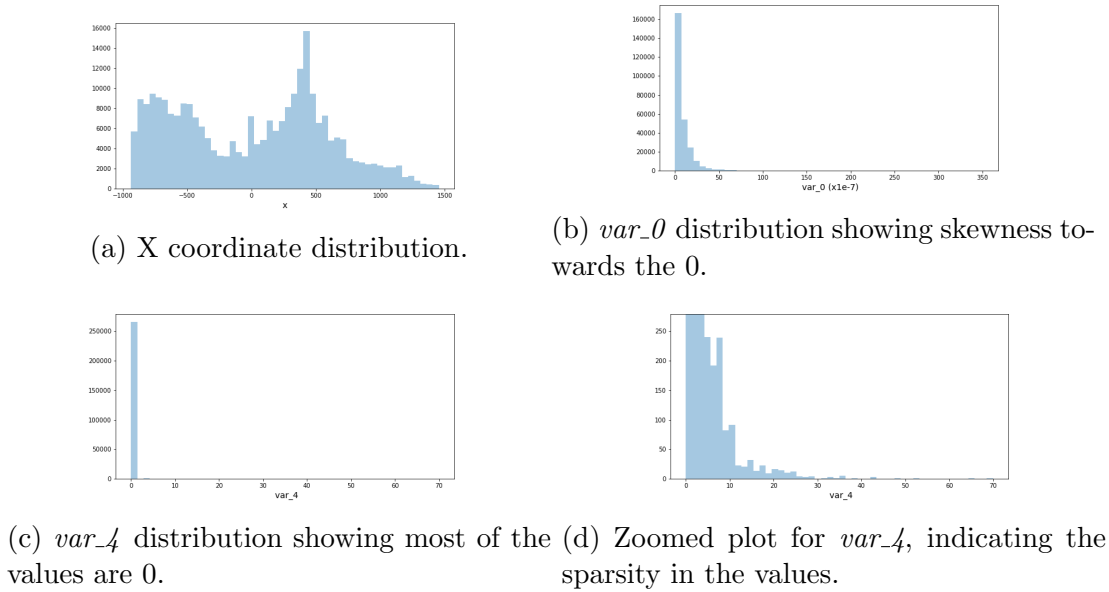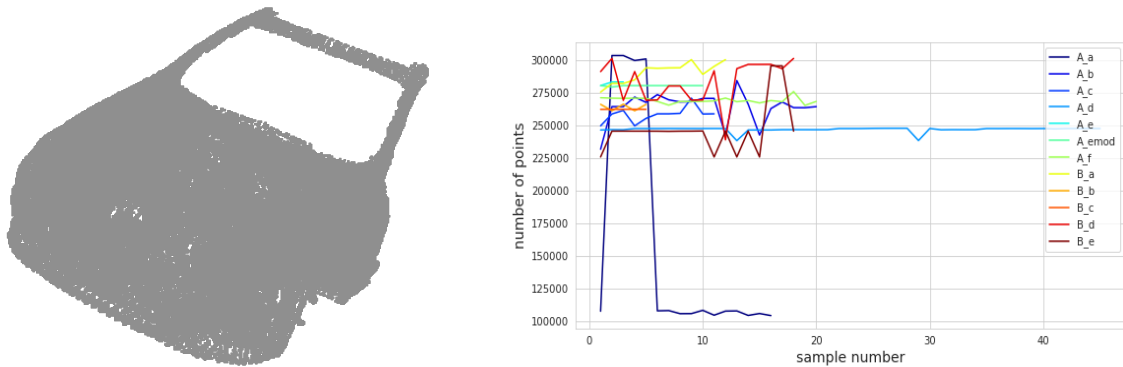
Figure 3: Distribution of the features for one car.

(a) Faulty point cloud: missing windshield. (b) Number of points in each point cloud.

Figure 4: Faulty point clouds in the data set 1.

remaining point clouds. We observe that some of the point clouds have some points which are not a part of the car, as shown in Figure 5a. To automatically detect and remove these points, we utilize the statistical outlier removal algorithm [25]. This algorithm computes the average distance of any point to its neighbors and marks a point as an outlier when it is far from its neighbor compared to the average point cloud. It then takes the "number of neighbors" (nn) and standard deviation ratio (std_ratio) as deciding parameters. The nn parameter defines the number of nearest points to consider while computing the average distance of each point. The std_ratio parameter sets the threshold based on the standard deviation of the average distances across the point cloud. Using this algorithm, we were able to detect the outlier points in all the given point clouds. Figure 6b shows the detected outlier for the point cloud shown in 5a, the detected outliers are highlighted by red color. Due to the very nature of the algorithm, increasing the nn or std_ratio parameter causes a reduction in the number of points detected as outliers. Since the algorithm uses the stan-
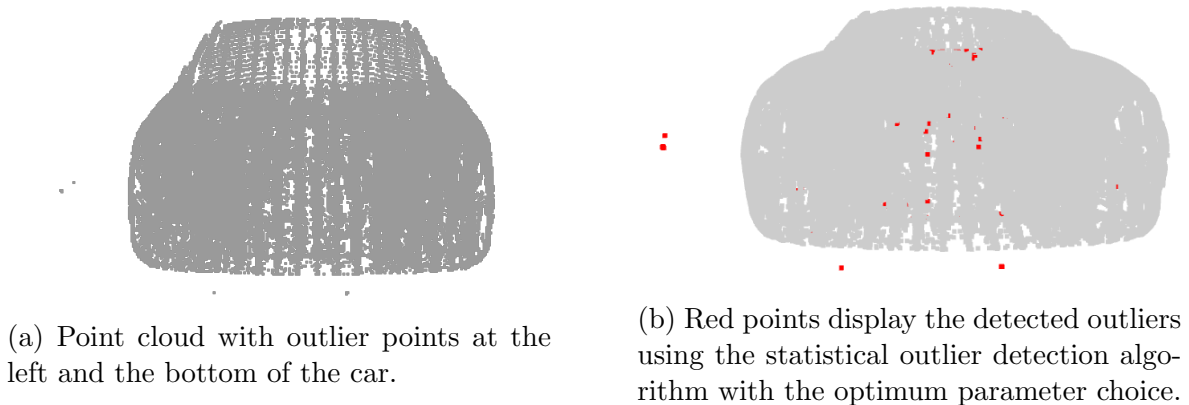
(a) Point cloud with outlier points at the left and the bottom of the car.

(b) Red points display the detected outliers using the statistical outlier detection algorithm with the optimum parameter choice.

Figure 5: Outlier treatment.

dard deviation ratio as the threshold, it can cause some of the inner points to be classified as an outlier when the sampling of points is not homogeneous or near the boundaries of the point cloud. Hence, our objective is to minimize the classification of inliers classified as outliers and maximize the classification of actual outliers as outliers.

To solve this problem, we perform a parameter space search across the std_ratio values and nn values. We select the lowest possible value of nn and std_ratio for which the change in the number of points detected as an outlier is sufficiently small. A plot for best values is shown in Figure 6, using these optimum parameters, the outliers were removed from all the point clouds.

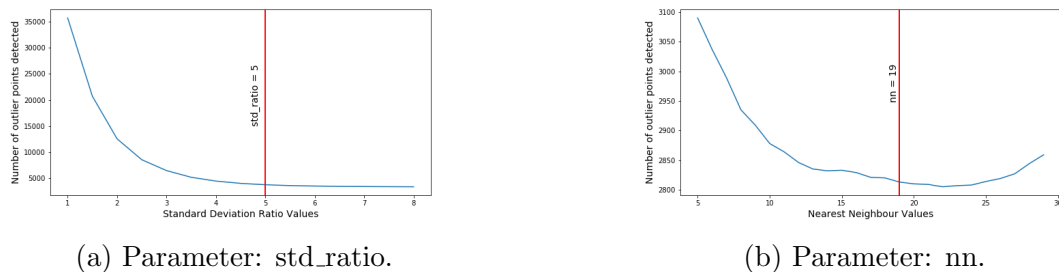(a) Parameter: std_ratio.

(b) Parameter: nn.

Figure 6: Parametric study for number of points classified as outlier.

The next preprocessing step is motivated by the idea that any rigid body transformation to the entire point cloud is not characterized as a change in the point cloud for this specific problem. Hence to nullify such changes in the point cloud, we try to detect and apply the rigid body transformation that would bring the two point clouds as close as possible. To achieve this goal, we employed the Iterative Closest Point (ICP) algorithm, which tries to minimize the distance between two point clouds[6]. ICP takes two point clouds, one as the source and second as the target, and tries to bring the source close to the target. Each iteration of ICP performs the following three steps:

- Find correspondence set $\mathcal{K} = (\mathbf{p}, \mathbf{q})$ from the target point cloud $\mathbf{P}$, and source point cloud $\mathbf{Q}$, using KDtree[14].

- Update the transformation matrix $\mathbf{T}$ by minimizing the objective function $E(\mathbf{T})$ described below.

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q}) \in \mathcal{K}} \|\mathbf{p} - \mathbf{T}\mathbf{q}\|^2. \tag{1}$$

- Apply transformation $\mathbf{T}$ on the source point cloud.

Since the first step of ICP is based on the computation of a correspondence set, it is usually recommended to provide an initial transformation, which can result in a larger and better correspondence set. The provided point clouds are denser in the front of the car (hood side) than the back (windshield side); we exploit this information in developing the initial transformation. First, we compute the minimum volume bounding box [3] for both source and target point cloud, as shown in Figure 7. For each bounding box, we find the point O, and using the position of O, we calculate initial translation transformation, further using the relative alignment of axis AB about the point O, we also detect the initial rotational transformation. These transformations are then applied to the source point cloud before performing the ICP.
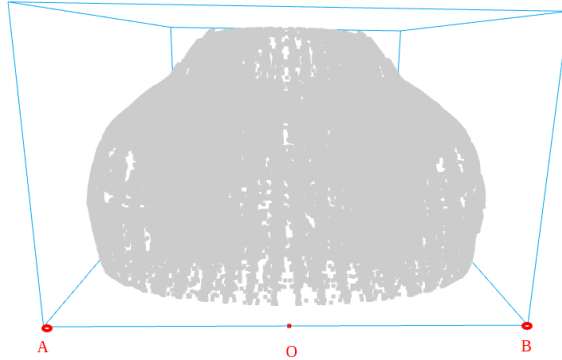

Figure 7: Minimum volume bounding box for the point cloud. The point O is used for detecting initial translation transformation while axis AB is used to detect the initial rotational transformation.

For change detection procedures used during our analysis, we are not interested in the windshield of the car since any changes to the front of the car will be of most influence for the HIC value of the car. During analysis of the car models small slices were taken of the car along the X-axis, the axis along which a car normally moves. Of these slices the maximum value along the z-axis, the vertical axis, among all points was measured. Here we observe a smooth line across the hood and another smooth line over the windshield, both with a different slope. This is plotted for 5 randomly selected cars in Figure 8a. At the point where this slope changes most, the *kink*, the first derivative makes a jump, see Figure 8b. This jump is used to find the point where the hood ends and the windshield begins. For the second derivative where this kink is also clearly visible as a spike, the reader is referred to Figure A.1. Using the position of this kink, the car model is separated into two parts: the windshield and the front of the car.
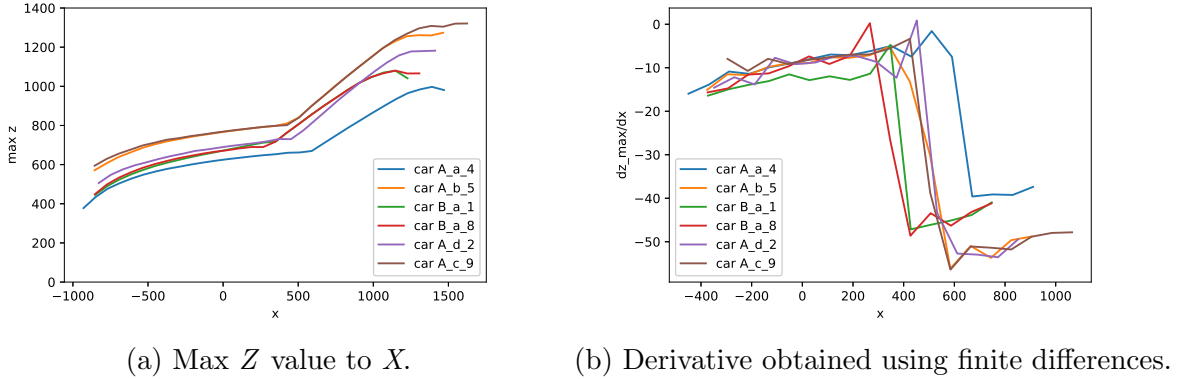
(a) Max Z value to X.   (b) Derivative obtained using finite differences.

Figure 8: Kink visualized for 5 randomly selected car.

3   State of the art

Based on the preprocessed data from the previous chapter, several state-of-the-art methods were explored for performing the comparison of the preprocessed point clouds. For point cloud registration, both rigid and non-rigid point cloud registration are introduced in this section. Afterwards, some point cloud segmentation approaches are discussed. Finally, one point cloud surface reconstruction method - alpha shape - is considered.

3.1   Point cloud registration

Point cloud registration is also known as point matching. It focuses on finding the correspondence between two point clouds. Typically, there are two types of point cloud registration: rigid and non rigid.

Rigid body point cloud registration investigates the transformation between two point clouds such as rotation and translation. One classical rigid point cloud registration is Iterative Closest Point (ICP) [6], which was applied by us for the preprocessing step. Another group of point cloud registration is the non-rigid one. Compared to the rigid, it can also be used to detect the deformation of the point cloud. Two non-rigid point cloud registration approaches are introduced.

Coherent Point Drift (CPD) can be used to describe the deformation of the two point clouds [15]. Through a Gaussian mixture model, the algorithm tries to fit the source point cloud to the target point cloud by maximizing the likelihood. The output of the CPD is the displacement field, which indicates the movement of the point cloud. The displacement field is defined as:

$$\vec{u} = \vec{R_1} - \vec{R_0} \tag{2}$$

where $\vec{R_1}$, $\vec{R_0}$ are the position of two point. $\vec{u}$ is the displacement vector. We can utilize the displacement matrix to detect the movement of the point in a very local manner.

Global and local mixture distance (GLMD) is another classical non-rigid point cloud registration [24]. It includes two steps: correspondence estimation and transformation

updating. For this algorithm, a local distance and a global distance are defined. The global distance is to measure the squared Euclidean distance between two point clouds.

$$G_{a_i b_j} = G(a_i, b_j) = \|a_i - b_j\|^2 \tag{3}$$

Local distance measures the similarities between point clouds and is defined by:

$$L_{a_i b_j} = L(a_i, b_j) = \sum_{k=1}^{K} \|N(a_i)_k + (b_j - a_i) - N(b_j)_k\|^2 \tag{4}$$

where K is the number of neighboring points, $N(a_i)_k$, $N(b_j)_k$ are the Kth closest point. Next, the two distances are combined to obtain a cost matrix. Based on this cost matrix, the correspondence between two point clouds is estimated by minimizing global or local structural differences.

3.2 Point cloud segmentation

Currently, one of the most notable method for supervised point cloud segmentation method is PointNet [19]. First published in 2016 by scholars from Stanford university, PointNet is a powerful neural network architecture. It can be used for a wide range of applications. With regard to this project, the ability of the PointNet architecture to segment the car is of most interest. The code for PointNet is open source and available under MIT license. In 2017, PointNet's successor PointNet++ was released [18]. The ShapeNet database [10] can be used for training the neural network, after which transfer learning could be used ant he network trained on our data. After having performed initial tests during the start-up phase of the project, it was decided not to investigate this direction further as the data set available to us is rather small. On top of that our data is unlabeled, leaving us with no data to train a neural network.

Another popular approach for point cloud analysis is clustering. Clustering is an unsupervised machine learning method that has been used since before the presence of computers. Clustering algorithms can offer a way of segmenting the cars into different clusters representing components. Using this the hood could potentially be extracted, which would allow more focus on comparing separate parts of the car and greatly reduce the size of the point clouds to be compared.

An interesting candidate for this purpose is the density-based spatial clustering of applications with noise (DBSCAN) algorithm [11]. One of the most interesting features of DBSCAN is that the number of clusters does not have to be provided. The algorithm determines how many clusters it forms. The hypothesis was that different components of the car are made of different materials, such that clustering on spatial information and material properties would lead to segmentation of different components. Not all variables could be included in the clustering algorithm as the data suffers from the curse of dimensionality causing all points to be far separated from one another [4]. This would result in either many different clusters, or one big cluster of all points.

Dimensionality reduction to improve the performance of the clustering has been investigated. Initially, feature selection has been tried but resulted in either hundreds of clusters,

or one big cluster. Next, the derivatives of some of the variables for each point were approximated using a simple form of finite differences. However, this engineered feature again did not give the desired results. The failure of this method is likely because the variables between components were not distant enough to allow for clear segmentation of these components.

3.3   Point cloud surface reconstruction

Alpha shape is defined as a surface mesh which covers the boundary of the point cloud in 3D space. The idea of the alpha shape is the following: given the finite point cloud S, we choose any two points $P1$, $P2$ inside the point cloud and draw the circle with the radius alpha. If the circle contains only $P1$ and $P2$, then $P1$ and $P2$ are recognized as boundary points, as shown on 9.
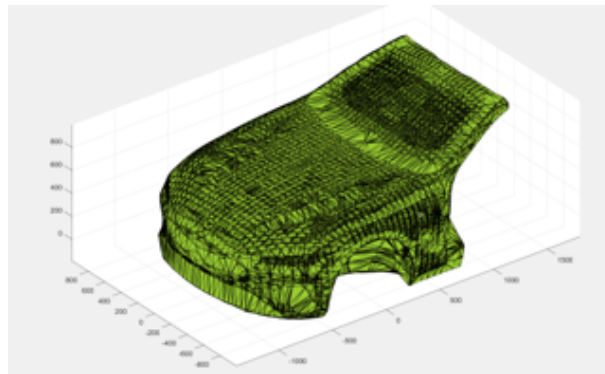

Figure 9: Alpha shape

Using alpha shape, we could numerically calculate the surface area and volume of the car. However, as a shape descriptor, alpha shape still has some drawbacks. Given a point cloud, it has a tendency create a surface that moves in between the points into the car. This adds unexpected inner surfaces to the surface calculation leading to inconsistent surface areas and volume calculated. To overcome these problems, we attempt to " "close" the point clouds on two planes where the problems were most prevalent. We project all the points onto the minimum Z plane to create an artificial layer which we merge with the point cloud. By adding such a layer, the bottom of the car is totally closed. Due to time constraints, the alpha shape was dropped in this project. It is a promising method to quantify the shape.

4   Applied Methods

After performing the previously described analysis and applying the preprocessing steps, we developed an algorithm to detect the changes in the point clouds. The change detection was split into two parts: high-level feature and low-level changes. We use high-level features to find the most similar car available in our database. Using this most similar car, we proceed with the low-level change detection. We are now able to assume similarity between the cars. The low-level changes would compute the changes at a granular level, between the closest car in the database and the provided car.

4.1   High level feature detection

Each of the provided point clouds contains geometric information in the form of X, Y, and Z coordinates; this geometric information can be used to extract some broader information about the data, like length, width, and height of the car. Further, the provided point cloud contains information on the mass of points that can be used to compute the total mass of the car. These features can be used to differentiate between different car models and families, in a broader sense, in the provided data set.

In addition to the features for the whole car, like total length and total height, we can also extract similar features for the hood and windshield using the segments of the car, which is described in section 2.2. For the high level analysis, we consider only 10 features, which are shown in Table A.2.

All the length, width, and height parameters were evaluated using a bounding box around the point cloud. The algorithm used for constructing the bounding box is described in section 4.1.1. Since some of these features are correlated to each other, we drop one of the two correlated features, a detailed study of this is provided in section 4.1.2. After the feature analysis, we do the prediction step where we find the closest matching point cloud to the given new point cloud, the analysis of this is provided in section 4.1.3.

4.1.1   Bounding Box

To compute the dimensions of a point cloud, we compute the vertices of a minimum volume bounding box around the point cloud. The basic version of the algorithm, which does that has a $\mathcal{O}(n^3)$ complexity [16], given the number of points in a point cloud and three bounding box computations per point cloud, therefore it becomes expensive to use this approach. So we employ the method described by Barequet G., et al. [3], where we compute the convex hull for the point cloud. And for each face of this convex hull, we construct a bounding box such that this face lies on one of the faces of the bounding box. Doing so, results in a finite number of bounding boxes, and the one with minimum volume is selected. Computing a convex hull has a complexity of $\mathcal{O}(n \log n)$, and each convex hull has a finite number of faces resulting in the overall complexity of $\mathcal{O}(n \log n)$ per bounding box.

Figure 10 shows two of the possible bounding boxes that can be computed based on the described method. We formulate all such bounding boxes and choose the one with the least volume. Using this bounding box, we calculate the total length, width, and height of the car. After this, we split the car into two parts separating the front and the rear parts of the car, using the method described in section 2.2. As shown in Figure 11, these boxes were used compute the dimensions of the hood and windshield. Table A.2 describes the meaning of each of these high level features.

4.1.2   Feature Analysis

After the computation of the high level features, we analyze these features to understand the trend across several point clouds. We employ these extracted features for predicting the closest car model, to a given point cloud, in the database. These features also describe the relative geometric changes between the two point clouds, like change in total length or hood length.
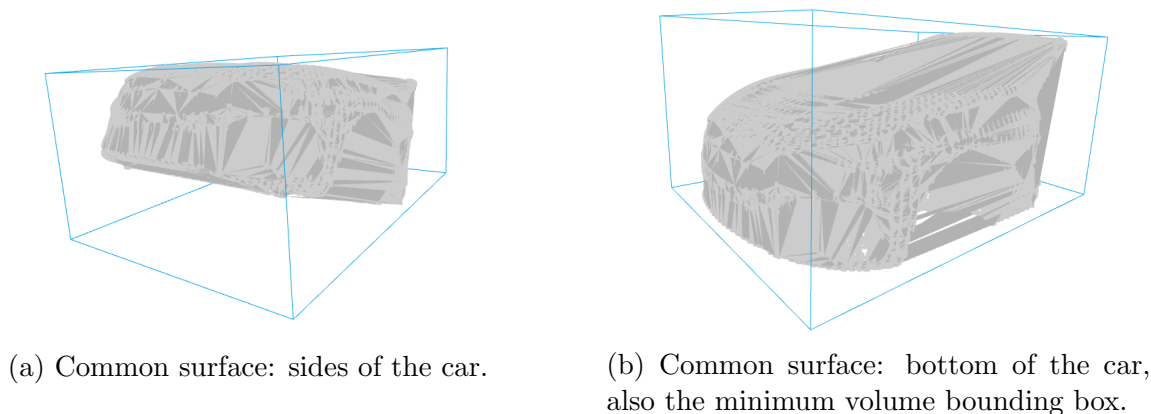
(a) Common surface: sides of the car.

(b) Common surface: bottom of the car, also the minimum volume bounding box.

Figure 10: Bounding Boxes having different common surface as with the convex hull formed on the point cloud.

(a) Minimum volume bounding box for the front section of the car.
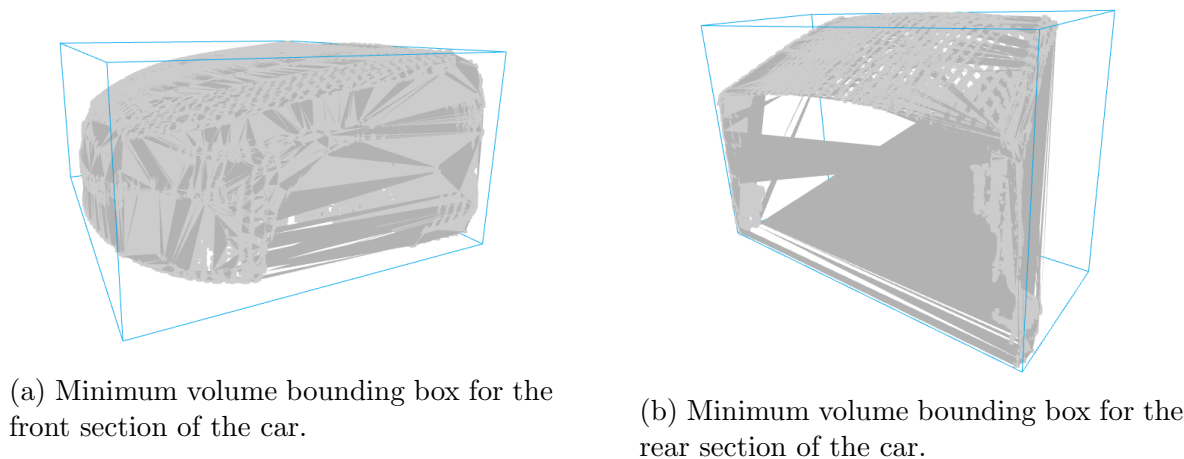
(b) Minimum volume bounding box for the rear section of the car.

Figure 11: Minimum volume bounding box formed on the split point cloud using the method described in section 2.2.

Feature analysis becomes critical for this specific problem since our first data set results in only 151 useful point clouds, each having 10 high level features. Therefore our training set is minimal for any machine learning classification algorithm. Further, as we are constrained by the total number of point clouds available, we try to reduce the number of training features. Reducing the number of training features results in a better generalization of our classification model, and avoids overfitting.

To achieve this goal, we start by training a base prediction model, using *Random Forest Classifier* [9], to predict the car model based on all 10 features. We try to reduce the number of features in the following three steps, and we train a prediction model after each step using a new set of features and evaluate its performance. To evaluate the performance of each model developed, we split our data in an 80:20 ratio for the train and test set. The misclassification rate was computed using the test set. Further, the quality of the model was evaluated using 5 fold cross-validation.

1. **Feature Selection**: In this step, we compute the Pearson correlation coefficient ($\rho_{X,Y}$) [22] to understand the pairwise correlation between different high level fea-

tures. For any pair of features, (X, Y), having a high correlation coefficient,$\rho_{X,Y}$, we remove one of the X, Y from the training features. Figure 12 shows the Pearson correlation coefficient for all the features. We classify a pair of features to be highly correlated, if the Pearson correlation coefficient, $abs(\rho_{X,Y}) \geq 0.9$, Figure 1 shows the selected features based on this criteria. The prediction model was trained using these features, and the results are shown in Table 2. We can observe that the misclassification rate reduces after the feature selection step. Additionally, the mean cross-validation score remains approximately the same. At the same time, the standard deviation in the cross-validation score decreases, indicating a slight improvement in the generalization of the model.
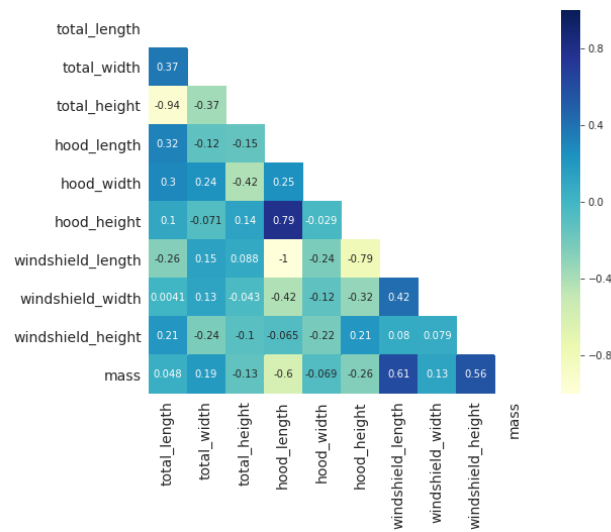

Figure 12: Pearson correlation coefficient matrix.

Table 1: Features selected after removing the highly correlated features.

Total Length	Total Width
Hood Length	Hood Width
Hood Height	Windshield Width
Windshield Height	Mass

2. **Principal Component Analysis (PCA)**[12]: In this step, we attempt to transform our features along the direction, which has a maximum variance, i.e., transform the feature to orient it along the eigenvectors. In doing so, we were able to reduce the number of features further. Figure 13 shows the cumulative explained variance plot. We can observe that 100% of data variance lies only in 6 dimensions, and hence we select these 6 principal components for training the prediction model. Table 2 shows the result of the application of PCA after the feature selection step. We observe that the misclassification rate has slightly increased. However, the cross-validation score of the model improves significantly with a smaller standard deviation value suggesting an improvement in the model.
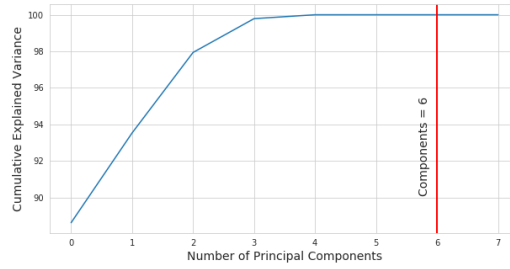
Figure 13: Cumulative Explained variance plot

3. **Feature Sensitivity**: In this step, we attempt to denoise the data by selecting the most sensitive features concerning output. For the sensitivity study, we employ ANOVA F-test [8], which for any feature checks relative variability across various car models with respect to variability in each model. The ANOVA test helps in determining which features are most informative in differentiating between the car models. Figure 14a shows ANOVA F-test results on the principal components. Based on the F-scores we select the optimum number of features (principal components) that generalizes the model best. Figure 14b shows that 5 out of 6 principal components generalize the predictive model best. Table 2 shows the results obtained with the selection of 5 best principal components. We can observe an increase in generalization, although the misclassification rate remains the same.
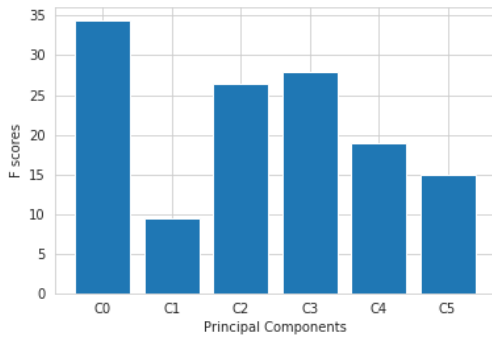
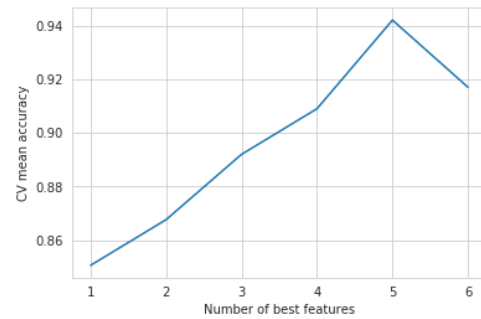(a) Anova F test result for the principal components.

(b) Cross validation score plot for $k = 1..6$ best features.

Figure 14: Feature Sensitivity study for denoising and best feature selection.

Table 2: Prediction model comparison for different cases described in section 4.1.2.

	Base Model	Feature Selection	PCA	Feature Sensitivity
Misclassification Rate	0.05769	0.03846	0.05769	0.05769
Cross Validation Score	0.909 ± 0.062	0.908 ± 0.057	0.934 ± 0.043	0.942 ± 0.034

Based on this analysis and the results shown in Table 2 we observe that we obtain the best generalization model when we apply all three steps, namely feature selection, PCA, and feature sensitivity. Therefore we decide to use these as preprocessing steps in our Car Matching algorithm when working with the Random Forest Model.

4.1.3 Car Matching

With the high level features and feature analysis in hand, we can employ several kinds of classification algorithms to predict the closest car model to a given point cloud. Considering the small data size and only 10 high level features, we can use algorithms like K-nearest neighbors (KNN) [2], which has $\mathcal{O}(n^2)$ complexity. Due to the nature of the high level feature and problem, KNN with slight modification turns out to be a very robust algorithm for this case. Nevertheless, we would like to develop a model that is suitable even when the data size is large. Therefore, we define our prediction model to switch from modified KNN to Random Forest Classifier [9] when the data size is sufficiently large. The modified KNN algorithm employs two steps for finding the best matching point cloud

1. Find K closest neighbors based on the higher-level features.

2. Return the point cloud, which has a minimum sum for the difference in the number of points and distance in feature space.

We employ this two-step method because we intend to find the closest car based on the physical features only. The second stage ensures that if there is more than one car that is similar in physical features, we choose the car, which is closest in terms of the physical feature as well as the number of points.

A similar approach was taken with the Random Forest Classifier. In the first step, we predict the closest car model. Then we search for the point cloud belonging to the predicted car model, which has a minimum difference in the number of points.

After finding the closest car from the database, the high level changes are registered, and the user can step forward to find low level changes which are described in section 4.2.

4.2 Low level changes detection

On this level we can assume similarity between cars. The following methods are aimed at finding the differences at finer level between two car models and might fail when the two cars differ greatly from one another. For the comparison the windshield is discarded, the process of discarding the windshield is described in 2.2. In this section, the *source* point cloud is used to describe the point cloud of a newly designed car to be compared to an already known car: called the *target*.

4.2.1 Slicing and slice correspondence

Each point cloud consists of many points. To detect changes at a finer lever, we want to reduce the dimensions of the point clouds being compared. We do this by cutting the point cloud in slices. These slices are made in the X-direction, which corresponds to the dimension of the length of the car. We slice in the direction of the windshield from the front of the car model. We do not expect a component to have moved drastically,

such that we can assume that the slices have a similar internal structure. We can do this as by the similarity assumption, the basic structure of all car models is the same. Now small changes, like a change of material or thickness of a component, can be detected. An example of the points belonging to one slice can be seen in Figure 15a. To prevent ending up with only a few points occupying the final slice a threshold was set up: in the last slice, if less than one-third of the slice width is populated with points, these points are pushed back into the previous slice. This is done to prevent comparing slices, where similarity can no longer be assumed. These parameters have been chosen after empirical tests.
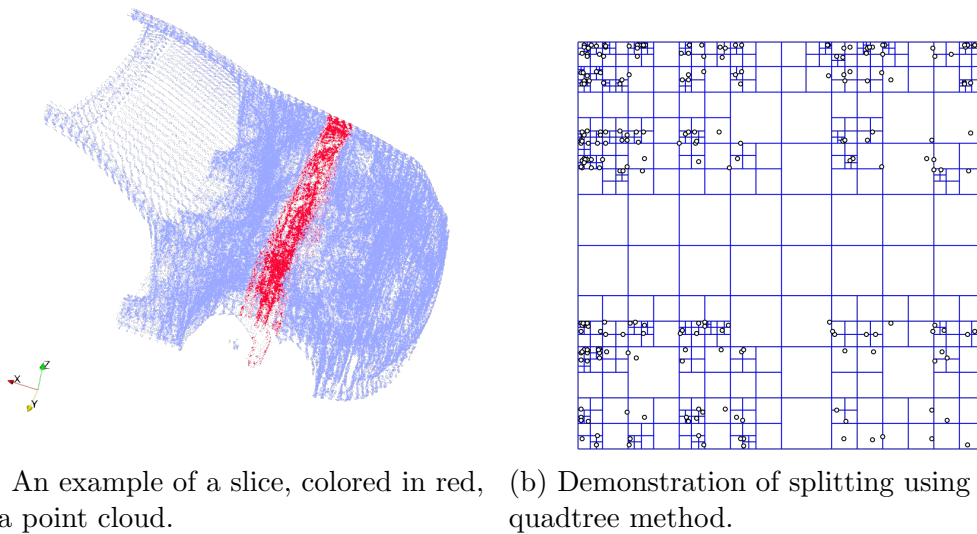
(a) An example of a slice, colored in red, of a point cloud.

(b) Demonstration of splitting using the quadtree method.

Figure 15: Two techniques used for dimensionality reduction.

The idea of slicing, as described above, works well as long as we have a target point cloud in the database, which has approximately the same total length as the source point cloud. However, a source point cloud which, for example, is an elongated car, could occur. In that case, there is no car in the database with similar length. We would like to construct a correspondence set using source slices and target slices, having the same slicing width, such that each member of the set is a pair of source and target slice. Further, each of these pairs has a similar geometric structure. In order to achieve this, we develop a slice correspondence algorithm, which is described in Algorithm A.1.

4.2.2   Dense variable comparison method

To compare dense variables, each point in the source is matched to the closest point in the target point cloud, its nearest neighbor in the target point cloud. To find the nearest neighbor, the Euclidean distance is used in two dimensions - Y, and Z - all points are projected to the slicing axis: the x plane. The dense variables generally describe material properties, thus matching points to their nearest neighbor makes intuitive sense. A similar internal structure of car and of the slice under study is assumed. By this assumption, each point will get a matched point of the same component in the target. Now, the material properties of each component can be compared.

A k-dimensional tree (k-d tree) is a data structure for efficiently querying a set of data points for a nearest neighbor, in Euclidean space, originally described in [5]. The k-d tree is a binary tree that represents a hierarchical subdivision of space. It works well for small k, especially compared to the number of points. If k approaches the magnitude of number of points in the data set, the data structure no longer leads to a better querying performance.

For our data set we are considering the Cartesian geometry of the points, with the x dimension projected to a plane, and hence only have $k = 2$ dimensions, with sample point clouds summing up to an order of 10^5 points. This makes the k-d tree applicable to perform fast nearest neighbour queries.

A rule needs to be prescribed for this subdivision of space: the splitting rule. In this project the sliding midpoint rule was used [14]. The construction of the k-d tree is performed in $\mathcal{O}(kn \log n)$, where k is the number of dimensions of the data and n the number of data points. Generally, the k-d tree is generated with a number of points at each leaf, and once that leaf is chosen as one of the candidate nearest neighbor, the final nearest neighbor is found using brute force. This is done to facilitate finding the p nearest points but this is not a requirement in this project, only one nearest neighbor is of interest. Therefore, a tree with one point at each leaf was used. Querying a k-d tree for a nearest neighbor can be done, on average, in $\mathcal{O}(\log n)$.

After having constructed the k-d tree using the target, we iterate over the source points and find a nearest neighbor for each point. This way the values of the variables can simply be subtracted, giving us a *delta*. The pseudocode of this procedure can be found in Algorithm A.3. In the ideal case, the number of points in both point clouds is equal and every point in the source has one corresponding point in the target.

In practise however, doubly matched points occur which can be caused by various reasons. It could be that the data generation of the point cloud is different in both samples leading to relative over- or under-sampling. Another contributor is when a geometric change has taken place between the source and target. In the areas where geometry has changed, an unusual matching of points could result.

4.2.3   Sparse variable comparison method

In the provided data set, the sparse features represent contact artifacts. These are discrete, integer values, which are non-zero only in a few small regions of a point cloud. Since these features contain zero value for most of the points, the first step towards the analysis is summing the value from all the points, for each variable, and comparing these in two point clouds. This gives a broad perspective of the change in these features. In this low level feature change analysis, we take yet another step in this direction and try to localize the changes in finer regions of the point cloud. Doing so helps us in pinpointing these changes to a small region, which can help an engineer working with these point clouds for their analysis.

To localize these changes, we extend the idea of summing these variables for the entire car to a small region in the car. We start this investigation at a slice level, where we split the slice in quarters and sum these variables for each quarter. If we detect any change for any quarter, then we further split that region recursively, until the maximum depth is reached. This method is commonly referred to as the quadtree approach [21]. For a

given point cloud, at this maximum depth level, for every sparse feature, we compute the sum of values for all the points present in that segment of the quadtree and compute the difference of values from the corresponding segment in the other point cloud. These values correspond to the local level changes. The algorithm used to execute the idea mentioned above is described in Algorithm A.3.

While the idea of not forming the quadtree recursively in the regions, where no change has been detected, saves the computational effort by avoiding redundant calculations, it might lead to problems when these values are shifted inside a quarter. Figure 16 shows one such example, as the quadtree would not be able to detect these changes at a coarser level, it would not perform any recursive split in this region. This limitation can be tackled to an extent, by modifying the Algorithm A.3 and introducing a thresholding parameter which enforces the recursive split until the thresholding depth.

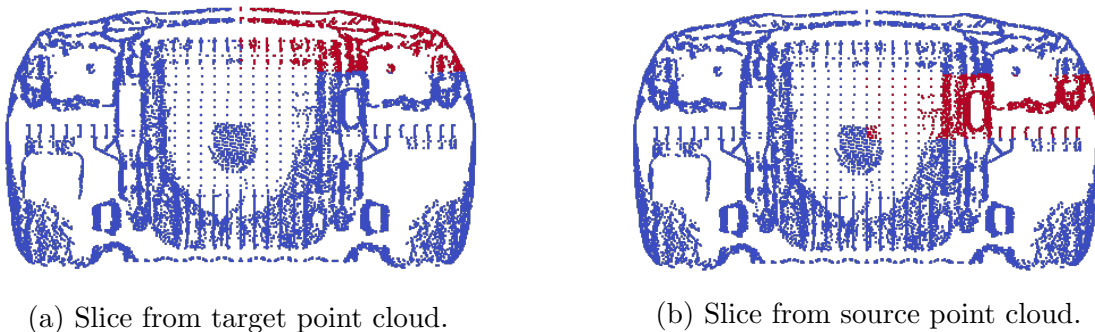(a) Slice from target point cloud.

(b) Slice from source point cloud.

Figure 16: Red points represent a non-zero value for a sparse variable in the slice. The blue points represent the zero values. The left image shows the target location of non-zero sparse values and right shows the source location of the non-zero sparse values.

The changes in the sum of sparse values can happen because of two reasons, first being the changes in the value of already existing points in both the point clouds (Type I change), the second being addition, removal, or both of new points in one or both of the point clouds (Type II change). The two changes require different treatments. The first kind can be handled by computing the difference in the sum of a sparse variable and assigning them to all the points having non-zero value for the corresponding variable. The second kind of change is dealt with by adding the missing points in the source point cloud with change values assigned to them and setting all other parameters to 0, which makes a clear indication that these missing points were added when looking at dense variables. Further, to aid in a localized study of the changes, we define a δpoint parameter for each sparse variable. The δpoint is the change in the number of points in a slice having non-zero value for the corresponding variable. These two kinds of changes are shown and discussed in section 5.

Similar to dense variable change detection, all the changes detected in the sparse variable were concatenated to the source point cloud.

4.2.4 Geometric comparison

To detect a change in the geometry, the Hausdorff distance between two point clouds is used. Given two sets of point A and B, the Hausdorff distance is defined by:

$$D_H(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\| \tag{5}$$

The Hausdorff distance is a directed distance metric, the general symmetric variant is simply defined by $\max(D_H(source, target), D_H(target, source))$. For computing the directed Hausdorff distance an implementation of the SciPy Python package [23] is used, based on [20].

Next, a similar approach as for the sparse variables is used. In the quadtree approach, another split is performed, if the Hausdorff distance is larger than a threshold, or the maximum depth is reached. The Hausdorff distance found on the maximum depth of the quadtree is assigned to all points in this split. If the quadtree traversal is prematurely stopped, because the distance found is smaller than the threshold, all points in the split get a zero Hausdorff distance. The pseudocode of this procedure can be found in Algorithm A.4. The dimension in which the hood has been sliced, is assumed to be constant for all points and thus not considered when calculating the Hausdorff distance. In the result, a high Hausdorff distance indicates geometric change, and a zero Hausdorff distance indicates no geometric change.

5 Results

Based on our analysis and developed algorithms for detecting high and low level changes, we developed a pipeline combining all intermediate steps to produce the final result. The pipeline incorporates all the preprocessing steps, high level feature extraction, closest car matching, and low level change detection.

5.1 Pipeline

The final product is a Python class that takes a point cloud and names of sparse and dense variable as an initialization parameter. It also provides the option to modify other default parameters which are used for the change detection algorithm. The user can call apply method with the location of the new car and gets an output file with all the changes written to it. Figure 17 shows the execution order of the algorithms described in section 4.

5.2 Car Matching

The car matching recommendation algorithm finds the best match from the database. Therefore, it can only be used after running the "fit" method in the pipeline. The fit method generates a compiled database of high-level features for all the existing point clouds. For a point cloud having about $3x10^5$ points, the higher-level feature detection takes about 40 seconds per point cloud, and the overall time required for finding the closest car is about 50 seconds.
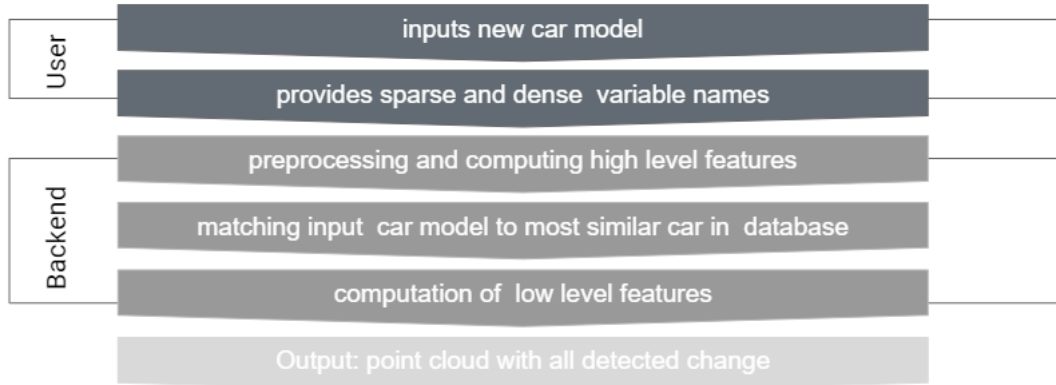
Figure 17: Diagram of the Pipeline.

To test the car matching algorithm, we make the following three test cases:

1. Data set 1 with one point cloud removed for testing.

2. Data set 2 with one point cloud removed for testing.

3. Data set 1 + data set 2 with one point cloud removed for testing.

For each test case, the "fit" method was called after the appropriate setup, and the database was updated. For test case 1, we make a total of 15 runs with a random selection of removed point cloud, for case 2 we make 9 runs with all the point clouds being removed once, for case 3 we make 15 runs with a random selection of removed point cloud.
In each case, the detected closest car belongs to the same car family and has the same car model as the test car. Further, it always finds the car model, which has about the same number of points in the point cloud. Hence the behavior of the algorithm is as per the expectation.

5.3 Low level changes detection

The dense, and sparse low level feature detection algorithm was run on the second data set with all variables: *var_0*, *var_1*, *var_2*, *var_3* and *var_6_tol_X*. *Var_0, ar_1*, and *var_2* correspond to the material properties. *Var_3* and all *var_tol_X* correspond to contact artifacts. The analysis takes approximately 15 minutes per car pair to complete. In this section, the scale of colouring has been changed to clarify the area of detection. A gray coloring is used when no significant change was found. Red and blue are used when there is a positive or negative change respectively. Validation of the low level change detection results was performed with the second data set provided.
The second data set was provided such to validate the model's results. All labels for the second data set were presented in Table A.4. The changes can be divided into three areas of change: the front hood outer skin, the front hood inner skin, and the front hood inner reinforcement. The results should show a change in the entire outer hood, inner hood

or in the front of the hood respectively. In some cars the variables representing contact artifacts had been changed. This should be detected on a low level indicating where the contact artifacts have been added or removed.

For the cars with a material change, as expected no geometric change was found. The maximum Hausdorff distance found is of, negligible, order of 10^-2 millimeter throughout the whole car. When inspecting the dense variable change for all dense variables we observe that the area associated with the respective change is highlighted. The detected change value is homogeneous, as expected when the material is changed. An example of this can be seen in 18b where the front hood inner skin's material was changed between car 1 and 6. The inner skin generally has a special structure to absorb an impact. This structure is clearly visible.

<div align="center">(a) Var_1.   (b) Hausdorff distance.</div>
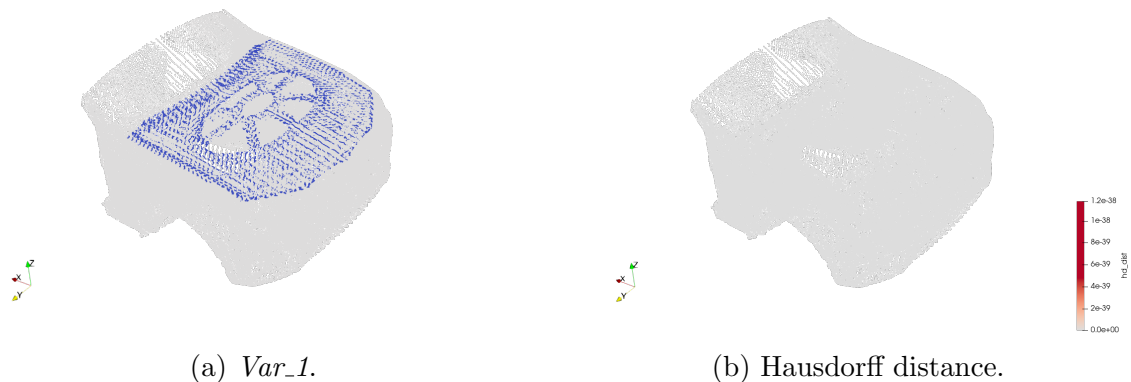
<div align="center">Figure 18: Change of car 6 with respect to reference.</div>

For the cars where the thickness of a component was changed, a geometric change in the correct areas was detected. Additionally, a dense variable change was also detected throughout the affected area. This result was not expected, further analysis showed this is likely due to the method used by BMW to extract the point cloud from the Finite Elements simulation. Both the dense variable and geometric changes detected can be seen in Figure 19.

When both the thickness and the material were changed, the algorithm again showed results that coincide with the expectation. A geometric and variable change is clearly identified at the correct location in the car. This can be seen in Figure 20, where the front hood inner reinforcement's material and thickness was changed. Consequently, we see a geometric and dense variable change at this location.

A great advantage of the current approach is the ability to detect both large and small changes. For example, between car 1 and 3 the material of the front hood outer skin was changed from steel to aluminum. A drastic change with respect to the material properties, so a large delta is expected for the dense variables. Indeed, as can be seen in histogram 21a, where the average value of the variable is depicted by the red vertical line, we see a relatively large delta. On the other hand, for small changes like between car 1 and 6 where only the type of steel used was changed. The algorithm identifies this change and depicts it as relatively small compared to the mean value of the material property as can be seen in figure 21b. The histogram does not depict all deltas on exactly one point equal
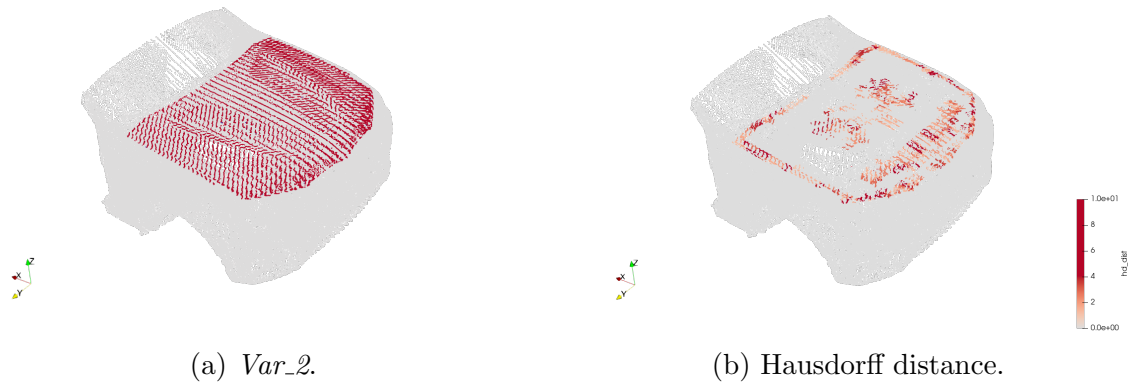
(a) *Var_2*.   (b) Hausdorff distance.

Figure 19: Change of car 4 with respect to reference.

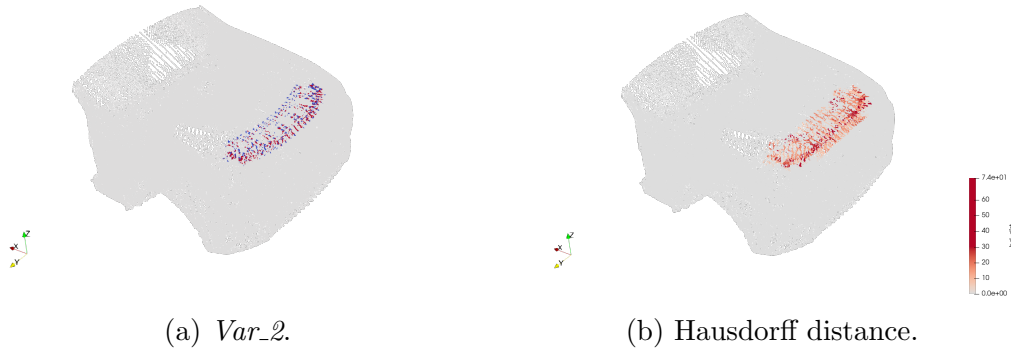(a) *Var_2*.   (b) Hausdorff distance.

Figure 20: Change of car 2 with respect to reference.

to the change in the material property between the two materials. This is again caused
by the data generation method, the change we see corresponds to the expected change.
We do see some outliers in the histogram. These outliers are caused by a match between
points of different components. In different components, materials with different proper-
ties are desired. The mismatched points have a large delta value. The mismatch happens
on the interface of a changed component. More points are mismatched when the geometry
changes between cars.

Looking at the distribution of the points with non-zero distance between matched points,
we get an idea of the quality of the matching. For car 6, plotted in figure 22b, no geometric
change is expected. The results shown in the diagram agree with what this expectation,
as a distance of 0.02 millimeter can be neglected. For car 4, plotted in figure 22a larger
values are found. This corresponds to the label that the thickness of a component has been
changed. The distribution suggests that the thickness change is of order 10^0 millimeter.

The validation case recommends observation of low-level changes in contact artifacts for
the cars 3, 4, 6, and 7. These changes were recommended in level and location, indicating
addition or removal of points or change in the value of sparse variable for the existing
points. Due to the very nature of these features, any changes in them do not cause
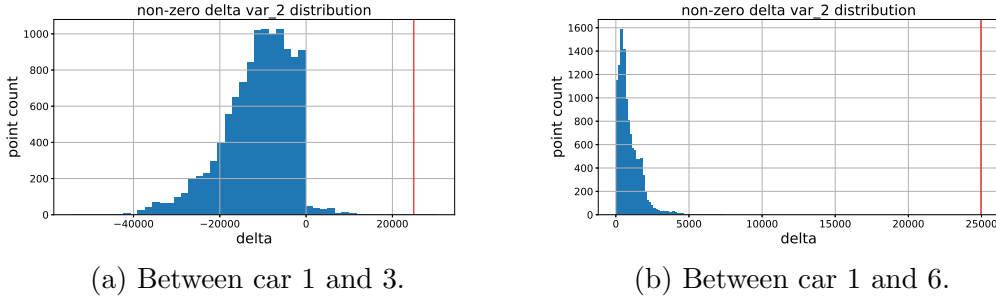a significant change in geometry. Mostly these sparse variable changes are confined to

(a) Between car 1 and 3.   (b) Between car 1 and 6.

Figure 21: Distribution of non-zero deltas found, red vertical line indicates the mean value of variable.

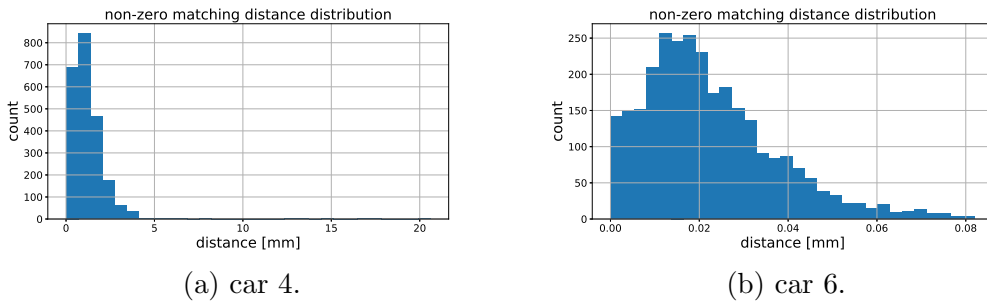(a) car 4.   (b) car 6.

Figure 22: Non-zero distance between matched points distribution.

several small regions in the point cloud.

Firstly, we consider the case where new sparse points were added to car 4. In this case, the changes detected through the low-level feature detection algorithm correspond to these added points in the car 4. The location of changes, as well as the magnitude of the change in that region, were also detected. Figure 23a show the detected changes for this car. Looking at Table A.1, we observe that for *var_3*, we detect only a Type II change with a negative amplitude, which corresponds to the addition of new points. Hence we can compute precisely the change with its magnitude and number of points added in case of Type II change.

The second case is detecting the case where the values of sparse variables have been changed. For this we consider the case of car 6 which has an asymmetric displacement in the value of the sparse variable near the windshield of the car, we observe that these changes were well detected, as can be seen in Figure 23b. Further, a look at the Table A.1 shows that these changes span over one slice since some of them are classified as Change Type II.

The categorization of changes in two categories helps us better define the changes on a local level without losing the global insight on the changes. Since these categorizations are done based on the local δpoint, and we can see that the global δpoint is 0, which implies that these changes involved displacement across the slices. There is also the possibility to extend these categorizations to accommodate wider possibilities, which can further enhance the overview of local changes.

The third case to consider is the mixture of the two cases described above, i.e., changing the value of sparse variables as well as adding/removing them from the point cloud. If

this occurs, we consider the variable *var_3* of the car 3, which has both the changes, i.e., the values of the existing point were changed as well as some points were removed. Table A.1 shows that the Change Type I, as well as Change Type II, have a positive value indicating removal of the points and reduction in the amplitude of existing points.
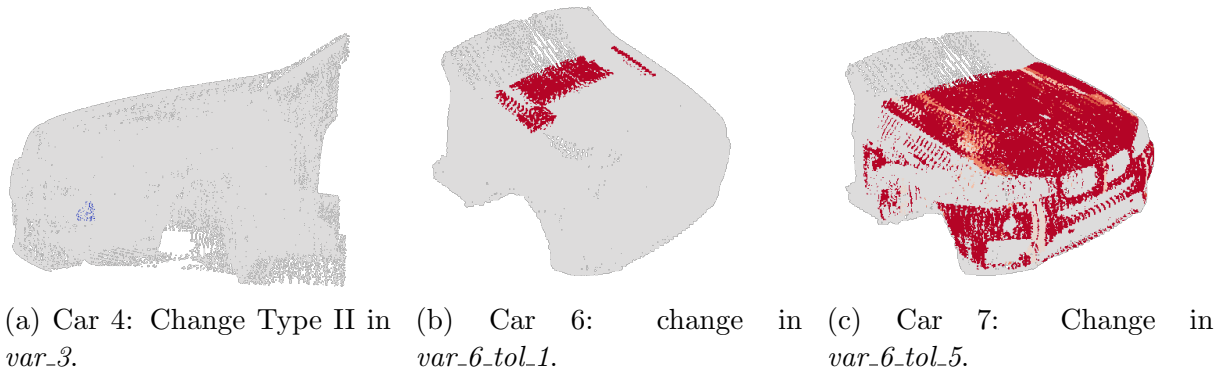

(a) Car 4: Change Type II in *var_3*.

(b) Car 6: change in *var_6_tol_1*.

(c) Car 7: Change in *var_6_tol_5*.

Figure 23: Detected Changes in the Sparse variables during the low level feature detection for Different Cars.

During the lower level feature detection analysis for the sparse variables, we observe that for some of the cars, which have no change label for the sparse variables in the list 1, sparse variable changes were detected. Such changes were observed only for the cars having geometric changes. Since these changes were not intentional, they were not present in the provided list. However, it can be seen that these changes are caused due to the movement of contact artifacts to accommodate geometric changes. Figure 23c shows the changes in the sparse variable induced by geometric changes.

Table A.1 displays a concise list of detected changes for all the point cloud in the data set two. The cars having no sparse variable changes were eliminated from the list.

6   Conclusions

In this project, we have developed an algorithm to detect the changes between two car models, which are presented in a human readable manner. The algorithm automates one of the components in the iterative design process that relied heavily on human expertise and experience. The comparison between cars can now be performed by AI that relies on data instead of expertise. The developed algorithm will improve as more data is being fed into it, making it more useful with time.

The algorithm exhibits the desired behavior. It detects both small and local changes, as well as big and global changes. These change detections are not only limited to the geometric changes but also non-geometric changes. The unexpected results that were found can be explained by the data generation. These are the limitations of the model that a user should be aware of.

This project has been validated on finding changes to a single component. In real use cases, more than one component can be changed. This can be solved by applying some density based clustering algorithm on the results, allowing the algorithm to analyze the changes separately. This is a promising direction for future research.

Additionally, the algorithm could prove to be the stepping stone for applying AI on a larger scale. The results of the comparison could be combined with the results of the FEM simulations to make a prediction on how a change of a component might impact the HIC value. This could open the way to eventual AI real-time recommendations to engineers. Potentially saving time for engineers doing monotonous tasks, and lives on the road.

Bibliography

[1]   2013. URL: https://www.euroncap.com/en/vehicle-safety/the-ratings-explained/vulnerable-road-user-vru-protection/head-impact/.

[2]   N. S. Altman. "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression". In: *The American Statistician* 46.3 (1992), pp. 175–185. DOI: 10.1080/00031305.1992.10475879. eprint: https://www.tandfonline.com/doi/pdf/10.1080/00031305.1992.10475879. URL: https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879.

[3]   Gill Barequet and Sariel Har-Peled. "Efficiently Approximating the Minimum-Volume Bounding Box of a Point Set in Three Dimensions". In: *Journal of Algorithms* 38 (Jan. 2001), pp. 91–109. DOI: 10.1006/jagm.2000.1127.

[4]   Richard Bellman. *Dynamic Programming*. Princeton University Press, Oct. 1957. ISBN: 069107951X. URL: https://www.xarg.org/ref/a/069107951X/.

[5]   JL Bentley. "Multidimensional Binary Search Trees Used for Associative Searching". In: *Communica-tions of ACM* 18.9 (1975), pp. 509–517. DOI: 10.1145/361002.361007.

[6]   P. J. Besl and N. D. McKay. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (Feb. 1992), pp. 239–256. ISSN: 1939-3539. DOI: 10.1109/34.121791.

[7]   Murray Bourne. *Head Injury Criterion (HIC) pt 2: HIC Index, example*. Jan. 2020. URL: https://www.intmath.com/applications-integration/hic-part2.php.

[8]   G. E. P. BOX. "NON-NORMALITY AND TESTS ON VARIANCES". In: *Biometrika* 40.3-4 (1953), pp. 318–335. DOI: 10.1093/biomet/40.3-4.318. URL: https://doi.org/10.1093/biomet/40.3-4.318.

[9]   Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[10]   Angel X. Chang et al. *ShapeNet: An Information-Rich 3D Model Repository*. Dec. 2015. arXiv: 1512.03012 [cs.GR].

[11]   Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: AAAI Press, 1996, pp. 226–231.

[12]   Harold Hotelling. "Relations Between Two Sets of Variates". In: *Biometrika* 28.3/4 (1936), pp. 321–377. ISSN: 00063444. URL: http://www.jstor.org/stable/2333955.

[13]   Jing Huang, Zhiying Liu, and Yongcheng Long. "A Numerical Investigation of a Novel Hood Design for PedestrianProtection". In: *The open mechanical Engineering Journal* 8 (2014), pp. 872–878. DOI: 10.2174/1874155x01408010872.

[14]   S. Maneewongvatana and Dave Mount. "It's okay to be skinny, if your friends are fat". In: *Center for Geometric Computing 4th Annual Workshop on Computational Geometry* (1999).

[15] Andriy Myronenko and Xubo Song. "Point-Set Registration: Coherent Point Drift". In: *arXiv e-prints* (May 2009). arXiv: `0905.2635 [cs.CV]`.

[16] Joseph O'Rourke. "Finding minimal enclosing boxes". In: *International Journal of Computer & Information Sciences* 14.3 (1985), pp. 183–199. ISSN: 1573-7640. DOI: `10.1007/BF00991005`. URL: `https://doi.org/10.1007/BF00991005`.

[17] *Pedestrian testing protocol.* Nov. 2011. URL: `https://cdn.euroncap.com/media/1463/euro-ncap-pedestrian-protocol-version-531.pdf`.

[18] Charles R. Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *arXiv e-prints* (June 2017). arXiv: `1706.02413 [cs.CV]`.

[19] Charles R. Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *arXiv e-prints* (Dec. 2016). arXiv: `1612.00593 [cs.CV]`.

[20] Abdel Aziz Taha and Allan Hanbury. "An Efficient Algorithm for Calculating the Exact Hausdorff Distance". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.11 (Nov. 2015), pp. 2153–2163. DOI: `10.1109/tpami.2015.2408351`. URL: `https://doi.org/10.1109/tpami.2015.2408351`.

[21] Yafu Tian et al. "A fast incremental map segmentation algorithm based on spectral clustering and quadtree". In: *Advances in Mechanical Engineering* 10 (Feb. 2018), p. 168781401876129. DOI: `10.1177/1687814018761296`.

[22] "VII. Note on regression and inheritance in the case of two parents". In: *Proceedings of the Royal Society of London* 58.347-352 (Dec. 1895), pp. 240–242. DOI: `10.1098/rspl.1895.0041`. URL: `https://doi.org/10.1098/rspl.1895.0041`.

[23] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* (2020). DOI: `https://doi.org/10.1038/s41592-019-0686-2`.

[24] Yang Yang, Sim Heng Ong, and Kelvin Weng Chiong Foong. "A robust global and local mixture distance based non-rigid point set registration". In: *Pattern Recognition* 48.1 (2015), pp. 156–173.

[25] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing". In: *arXiv:1801.09847* (2018).
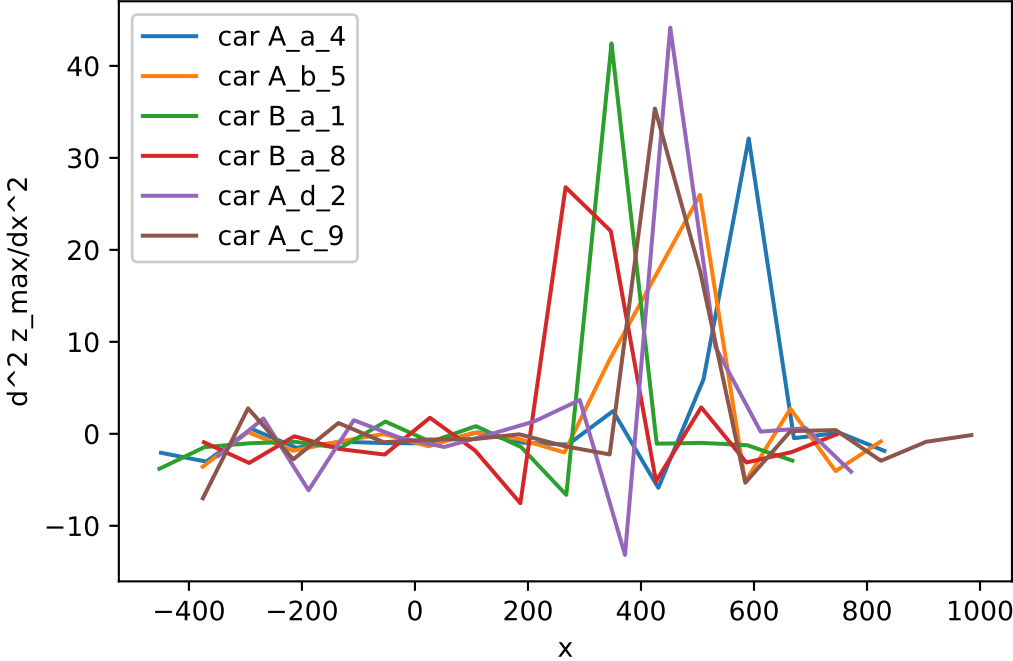
A   Appendix

Figure A.1: Second derivative, found using finite differences, of the line plotted in figure 8a.

Data: Source Point Cloud, Target Point Cloud, Slicing Width
Result: {(Source Slice, Target Slice)}
1 Compute the number of source and target slices.;
2 source start ← source$_{min}$;
3 source end ← source$_{max}$;
4 target start ← target$_{min}$;
5 target end ← target$_{max}$;
6 **while** *target start ≤ target end* **do**
7   Make target slice;
8   best fitness ← 0;
9   **while** *source start ≤ local search space* **do**
10   Make source slice;
11   compute fitness;
12   **if** *new fitness ≥ best fitness* **then**
13   best slice ← source slice;
14   best start ← end of best slice;
15   **else**
16   continue;
17   **end**
18   **end**
19   add (best slice, target slice) to correspondence set;
20   set source start to best start;
21 **end**

Algorithm A.1: Slice Correspondence Algorithm

**Data:** Slice correspondence set
Result: Mapping of form (points, delta)
1 delta mapping ← [];
2 **while** *slice correspondence set is not empty* **do**
3   source slice, target slice ← pop(slice correspondence set);
4   tree ← build_kdtree(target slice);
5   **while** *source slice point set is not empty* **do**
6   source points ← pop(source slice point set);
7   dist, nearest target point ← tree.query(source point);
8   delta ← nearest target point.variables - source point.variables ;
9   delta mapping ← (point, delta);
10   **end**
11 **end**

Algorithm A.2: Dense variable change detection Algorithm

Data: Slice correspondence set, maximum depth
Result: list of detected changes
1 list of changes ← [];
2 **while** *slice correspondence set is not empty* **do**
3   source slice, target slice ← pop(slice correspondence set);
4   stack ← make quadtree(source slice, target slice);
5   **while** *stack is not empty* **do**
6   source split, target split, depth = pop(stack);
7   source sum ← compute sum of all points for each feature for source split;
8   target sum ← compute sum of all points for each feature for target split;
9   **if** *source sum ≠ target sum for any sparse feature* **then**
10   **if** *depth == maximum depth* **then**
11   add changes to list of changes;
12   **else**
13   new split ← make quadtree(source split, target split);
14   stack.push(new split);
15   **end**
16   **else**
17   continue;
18   **end**
19   **end**
20 **end**

Algorithm A.3: Sparse variable change detection Algorithm

Data: Slice correspondence set, maximum depth
Result: Mapping of form (points, distance)

1 delta mapping ← [];
2 **while** *slice correspondence set is not empty* **do**
3   source slice, target slice ← pop(slice correspondence set);
4   stack ← make quadtree(source slice, target slice);
5   **while** *stack is not empty* **do**
6   source split, target split, depth = pop(stack);
7   distance ← compute_hausdorff_distance(source split, target split);
8   **if** *distance < threshold or depth >= maximum depth* **then**
9   **if** *distance < threshold* **then**
10   distance ← 0;
11   **end**
12   delta mapping ← (points in source split, distance);
13   **else**
14   new split ← make quadtree(source split, target split);
15   stack.push(new split);
16   **end**
17   continue;
18   **end**
19 **end**

Algorithm A.4: Geometric change detection Algorithm
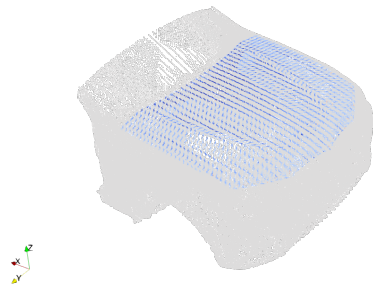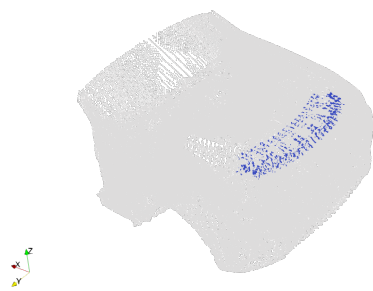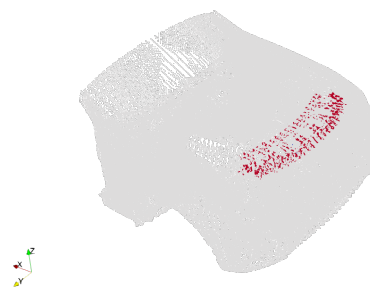
(a) Car 3 - *var_1*.
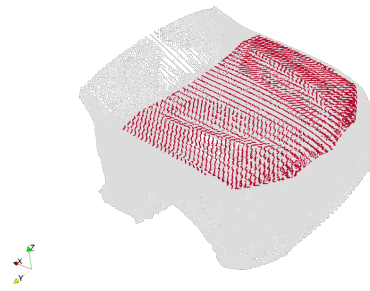
(b) Car 5 - *var_1*.
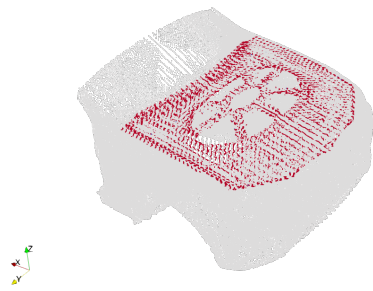
(c) Car 7 - *var_1*.
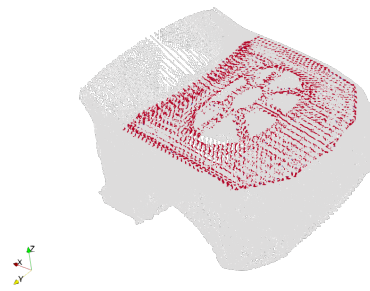
(d) Car 8 - *var_2*.

(e) Car 9 - *var_1*.
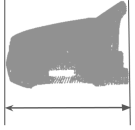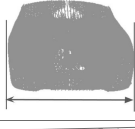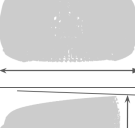
(f) Car 10 - *var_1*.

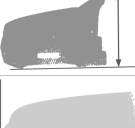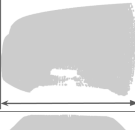Figure A.2: Low level change between cars of second data set and reference, all dense variables show similar results.
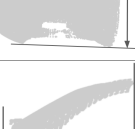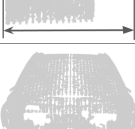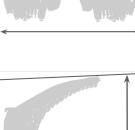
Table A.1: Detected changes for Sparse values for data set 2. A positive value for change correspond to removal of points or reduction in value, and a negative change correspond to addition of new point or increase in values. The classification of type of changes is provided in section 4.2.3

Car			Changes		
2	Variable name	Change Type I	Change Type II	overall change	$\delta point$
	var_6_tol_1	0.0	25920.0	25920.0	1083.0
	var_6_tol_3	0.0	12960.0	12960.0	1083.0
	var_6_tol_5	0.0	71280.0	71280.0	1083.0
	var_6_tol_10	0.0	12960.0	12960.0	1083.0
3	Variable name	Change Type I	Change Type II	overall change	$\delta point$
	var_3	276.0	540.0	816.0	10.0
	var_6_tol_1	-56.0	84.0	28.0	70.0
	var_6_tol_3	-28.0	42.0	14.0	70.0
	var_6_tol_5	-163.0	242.0	79.0	70.0
	var_6_tol_10	-28.0	42.0	14.0	70.0
4	Variable name	Change Type I	Change Type II	overall change	$\delta point$
	var_3	0.0	-161.0	-161.0	140.0
	var_6_tol_1	-212.0	256.0	44.0	39.0
	var_6_tol_3	-106.0	128.0	22.0	39.0
	var_6_tol_5	-593.0	720.0	127.0	39.0
	var_6_tol_10	-106.0	128.0	22.0	39.0
5	Variable name	Change Type I	Change Type II	overall change	$\delta point$
	var_6_tol_1	-24.0	40.0	16.0	54.0
	var_6_tol_3	-12.0	20.0	8.0	54.0
	var_6_tol_5	-72.0	117.0	45.0	54.0
	var_6_tol_10	-12.0	20.0	8.0	54.0
6	Variable name	Change Type I	Change Type II	overall change	$\delta point$
	var_6_tol_1	24.0	-8.0	16.0	74.0
	var_6_tol_3	12.0	-4.0	8.0	74.0
	var_6_tol_5	69.0	-24.0	45.0	74.0
	var_6_tol_10	12.0	-4.0	8.0	74.0
7	Variable name	Change Type I	Change Type II	overall change	$\delta point$
	var_6_tol_2	0.0	-7925.0	-7925.0	-3483.0
	var_6_tol_7	0.0	-2111.0	-2111.0	-1016.0
	var_6_tol_15	0.0	-873444.0	-873444.0	-136188.0
8	Variable name	Change Type I	Change Type II	overall change	$\delta point$
	var_6_tol_1	90.0	-46.0	44.0	61.0
	var_6_tol_3	45.0	-23.0	22.0	61.0
	var_6_tol_5	262.0	-135.0	127.0	61.0
	var_6_tol_10	45.0	-23.0	22.0	61.0

Table A.2: The table explains each of the higher level feature used in this work. The visual column shows an arrow in each image, which indicates the edge along which the dimension was measured.

Feature Name	Description	Visual
Total Length	Total length of the car	
Total Width	Total width of the car	
Total Height	Total height of the car	
Hood Length	Length of the front section of car containing hood.	
Hood Width	Width of the front section of car containing hood.	
Hood Height	Height of the front section of car containing hood.	
Windshield Length	Length of the rear section of car containing windshield.	
Windshield Width	Width of the rear section of car containing windshield.	
Windshield Height	Height of the rear section of car containing windshield.	
Total mass	Total mass of the point cloud.	

Variables	Dense feature	Sparse feature
X, Y, Z	×	
var_0	×	
var_1	×	
var_2	×	
var_3		×
var_4*		×
var_5*		×
var_6**		×

Table A.3: Variable categorization. (*) indicates that the variable is only present in data set 1. (**) indicates that the variable is only present in data set 2.

Car number	Changes applied in comparison to car 1
2	Material and thickness of front hood inner reinforcement
3	Material of front hood outer skin; location/level of var_3
4	Thickness of front hood outer skin; addition of new items for var_3
5	Material and thickness of front hood outer skin
6	Material of front hood inner skin; location/level of var_6
7	Thickness of front hood inner skin: location/level of var_6
8	Material thickness of front hood inner skin
9	Material of front hood inner reinforcement
10	Thickness of front hood inner reinforcement

Table A.4: changes applied to reference car in data set 2