Master's Degree in Industrial Engineering

Master's final project

Crash Pulse Prediction applied to Frontal Crash Configurations

Author
Pablo Mira Arana

Supervised by
Francisco José López Valdés

Madrid
June 2021

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Crash Pulse Prediction applied to Frontal Crash Configurations

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2020/21 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Pablo Mira Arana Fecha: 14/ 07/ 2021

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

7/14/2021

X

Francisco J. Lopez Valdes

Signed by: LOPEZ VALDES FRANCISCO JOSE - 09437440B

Fdo.: Francisco J. López Valdés Fecha: Fecha: 14/ 07/ 2021

Master's Degree in Industrial Engineering

Master's final project

Crash Pulse Prediction applied to Frontal Crash Configurations

Author
Pablo Mira Arana

Supervised by
Francisco José López Valdés

Madrid
June 2021

CRASH PULSE PREDICTION APPLIED TO FRONTAL CRASH CONFIGURATIONS

Author: Mira Arana, Pablo
Director: López Valdés, Francisco José
Collaborating Entity: Volvo Cars AB

PROJECT SUMMARY

1. Introduction

Occupant safety is one of the main challenges when designing a vehicle. As the number of vehicles increases every year around the world, enhancing their performance in a crash regarding the protection of the occupants is key to reduce the number of injuries and deaths. In these terms, studying what happens during a crash event is crucial for safety evaluation and improvement. To do so, numerous physical and virtual crash tests are performed, trying different scenarios and configurations. Both methodologies have their limitations, being the time consumption and high expenses the main ones.

Introducing Machine Learning to this field can bring numerous benefits, taking advantage of the existence of years of performed and recorded tests. The principal aim of this project is to help the evaluation of the movement response the vehicle has during the crash. This response is called Crash Pulse, and its importance and effect on the severity of the crash has been amply studied and proven.[1][2][3][4]

The overall behavior of the vehicle during a crash may vary on a large scale depending on how the crash occurs. Many crash types can be stated, but the main one is the division between lateral and frontal crashes. As for the first one, the intrusions in the cabin and lateral accelerations have large importance, the second one is largely determined by the decelerations suffered. This project will be focused on frontal crashes.

The objective of this project is to develop an Artificial Intelligence algorithm that is capable of predicting the resulting Crash Pulse with a given Crash Configuration. The Crash Configuration definition that will be used is defined in [5] and later on used in [6].

2. Methodology

The project has two main stages. The first one is the process of Data Analysis, where all the information from past tests is recorded, filtered, and pre-processed to be introduced to the AI algorithm. The second one is the design of this AI algorithm, its training, testing, and evaluation.

2.1 Data Analysis

The data analysis process is performed to create a valid dataset that can be understood by the AI model. The process to do so is described in Figure 1.

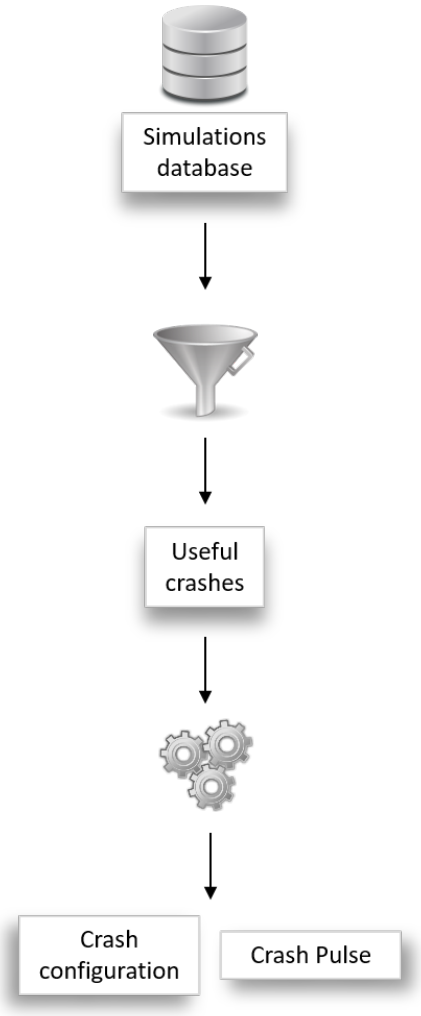

Figure 1: Data gathering process.

- **Selection**. From all the information available, the useful and pertinent simulations for this project must be selected. This includes car-to-car and car-to-moving barrier simulations, with a minimum length of recorded Crash Pulse and within the limits established for frontal and rear-end crashes.

- **Pre-processing**. The data needed for the AI implementation must be extracted from the selected simulations. This information extraction can be divided into two differentiated parts:

  – *Crash Configuration*. This is the input for the AI model. The parameters describing the crash must be computed from the raw simulation information. The information in files **d3plot** and **nodout** has been used to detect each object colliding using a clustering process, compute their velocity (HS and OS), their masses (HM and OM), and the angles describing their relative position. These angles are Host Collision Point Angle (HCPA), Opponent Collision Point Angle (OCPA), and Opponent Yaw Angle (OYA). The first two are referred to the First Point of Contact (FPOC). To be able to compare configurations with dimensional differences, normalization must be performed, to reduce the vehicle shape to a unit square.
  A graphic description is represented in Figures 2 and 3, and an example of the resulting Crash Configuration is represented in Table 1.
  The limit established for frontal/rear-end crashes is [-45,45] and [135,225] regarding HCPA.

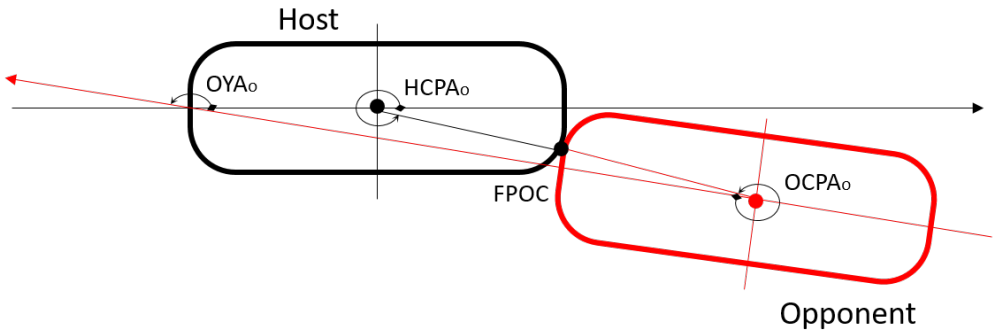

Figure 2: Crash Configuration Angles [5]

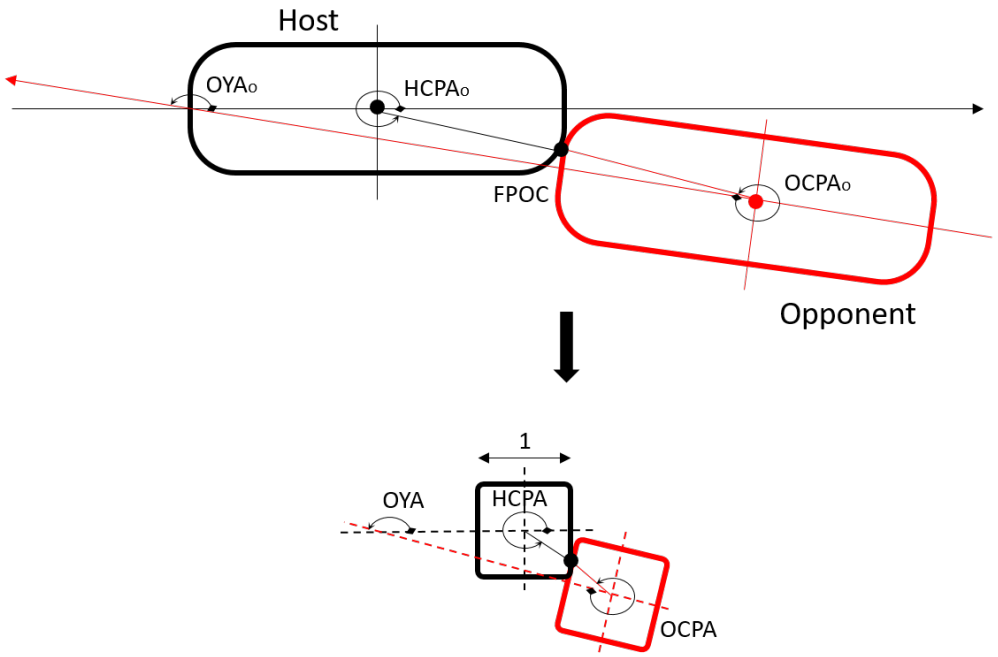

Figure 3: Crash Configuration Normalization [5]

HCPA	-38º
OCPA	-26º
OYA	160º
Host speed	13.5 m/s
Opponent speed	21 m/s
Host mass	2178.9 kg
Opponent mass	2335.4 kg

Table 1: Crash Configuration parameters

– *Crash Pulse*. This is the output of the model. The information of the crash pulse is extracted using a pre-existing script, that computes the crash pulse in terms of velocities. The crash pulse is formed by 6 time series, 3 axis translations and 3 axis rotations. For translations, the information has been converted to acceleration. Meanwhile, the rotations have been converted to angles. Each time series has been cropped to 80 ms and described using 16 timesteps. The resulting output information describing the crash pulse of each simulation is formed by 96 uni-dimensional points.

2.1 AI Modelling

The data has been split into 2 different datasets. The first one will be introduced to the model with input and output information, so it can learn the expected output. This set is called training set. The second one will be introduced only with input information, so the model will make blind predictions that can be then compared with the real output. This set is called test set.

As the output parameter is a continuous value, the prediction model will be regression. Four different algorithms have been tested. These are Neural Networks, Support Vector Machine Regression, Gradient Boost Regression, and Random Forest Regression. The first two algorithms are Kernel-based, as the last two are Decision Trees-based. Each algorithm has been modeled individually to obtain the best predictions possible.

3. Results

The resulting dataset is formed by 284 data points, 232 (81.7% of the dataset) for the training set and 52 (18.3% of the dataset) for the test set. This split is represented in Figure 4, where the dataset is represented in terms of HCPA (x-axis) and OCPA (y-axis) distribution.

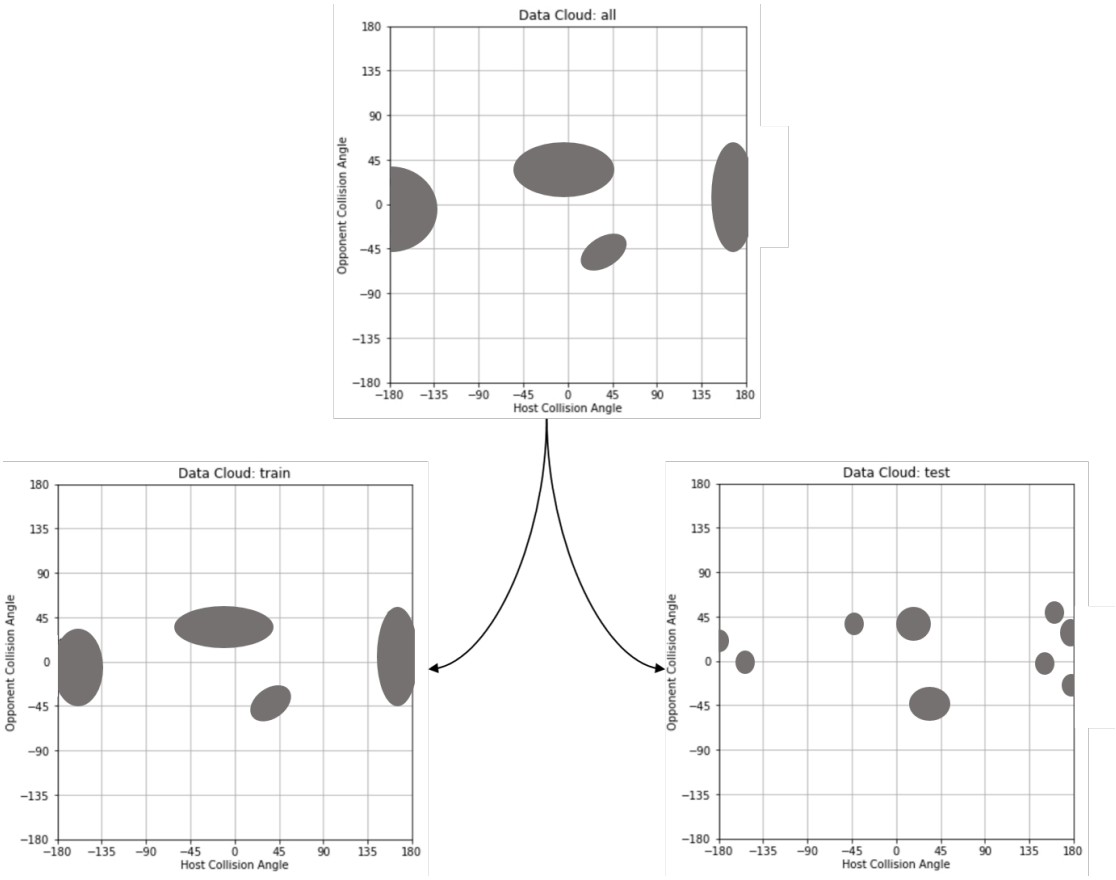

Figure 4: Resulting data cloud and splitting

The four models have been tested using R^2 and RMSE metrics, both for each step/prediction and for the complete translations/rotations structure. The average scores are represented in Table 2, and Figures 5, 6.

	Translations		Rotations	
Algorithm	RMSE	R^2	RMSE	R^2
Neural Network	0.020	0.728	0.0024	0.103
Support Vector Machine	0.017	0.805	0.0023	0.319
Gradient Boost	0.019	0.747	0.000126	0.834
Random Forest	0.0189	0.751	0.00229	0.937

Table 2: Average error metrics for each model

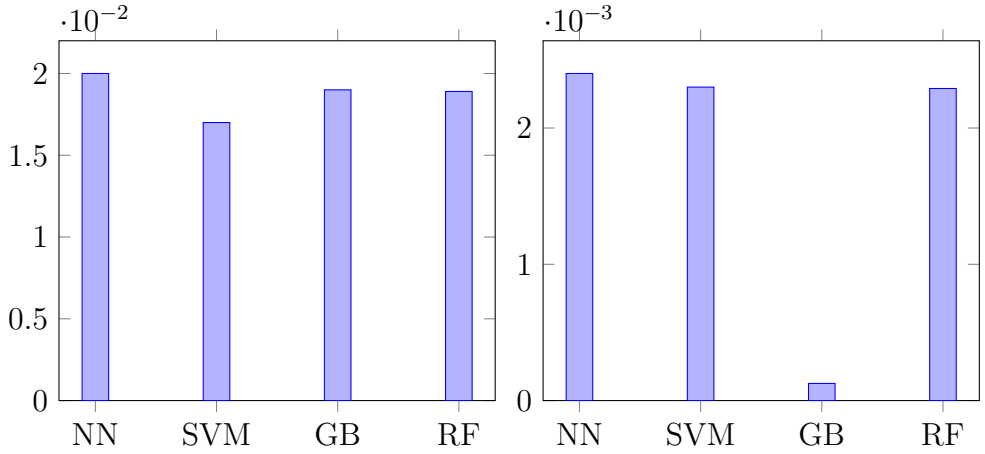

Figure 5: Comparison of RMSE for translations (left) and rotations (right)

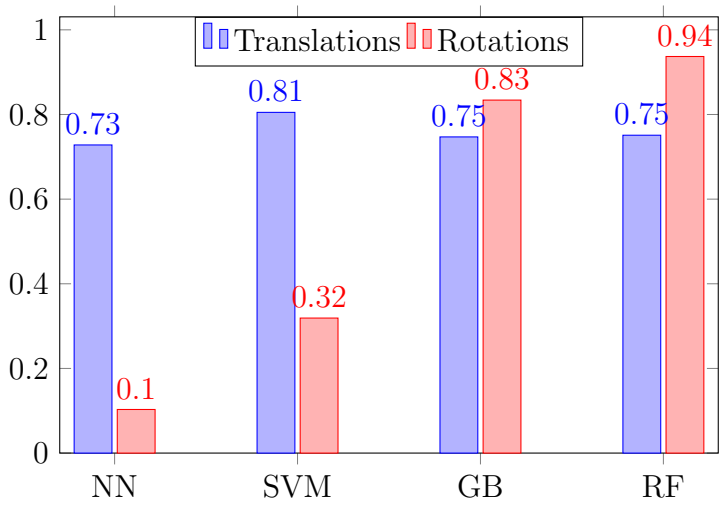

Figure 6: Comparison of R^2

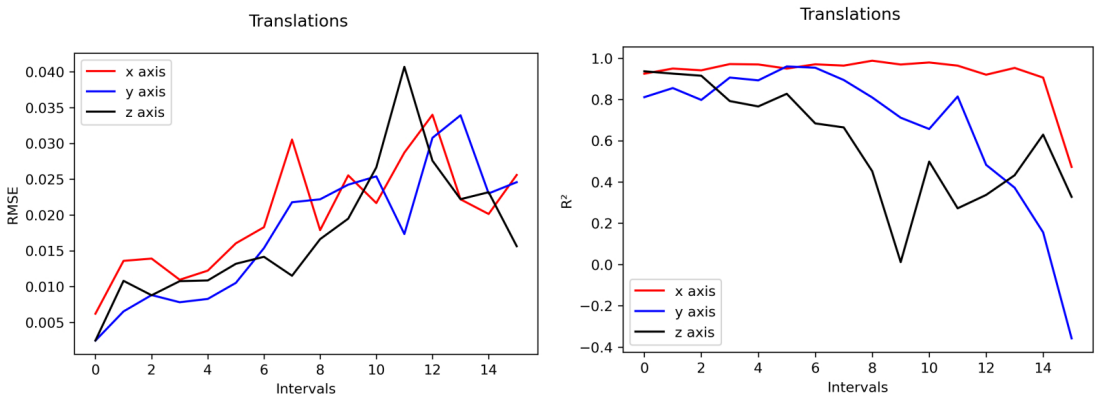

Figure 7: RMSE and R^2 values for each step in Neural Network model for translations

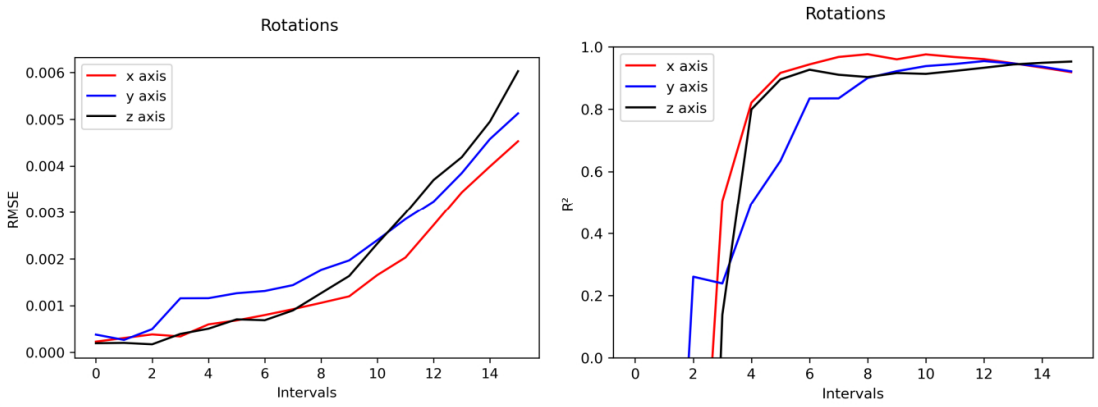

Figure 8: RMSE and R^2 values for each step in Neural Network model for rotations

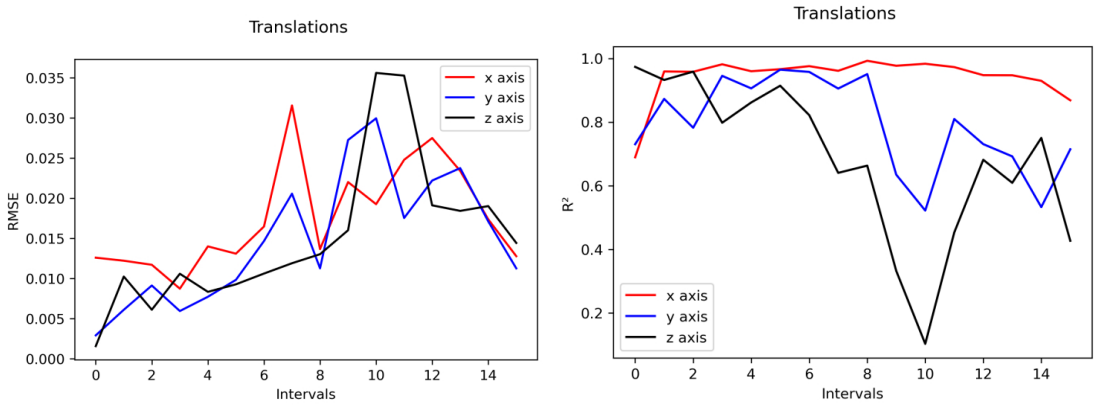

Figure 9: RMSE and R^2 values for each step in Support Vector Machine model for translations

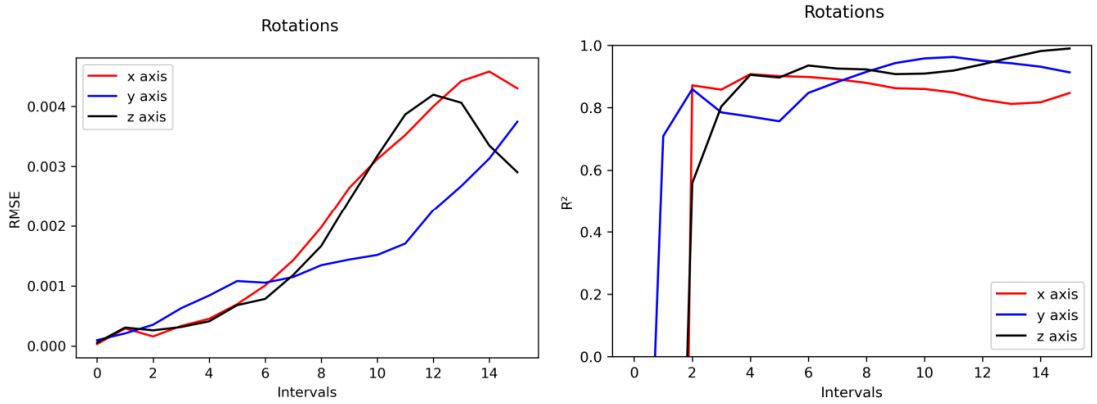

Figure 10: RMSE and R^2 values for each step in Support Vector Machine model for rotations

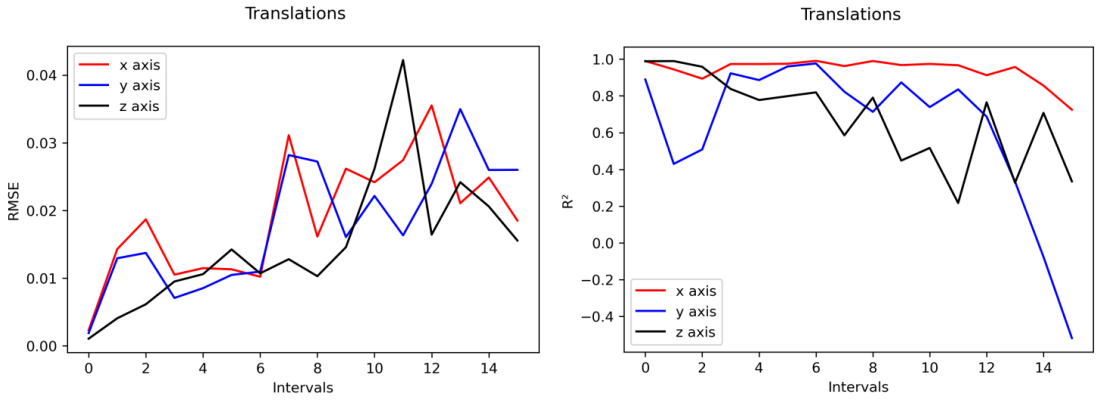

Figure 11: RMSE and R^2 values for each step in Gradient Boost model for translations

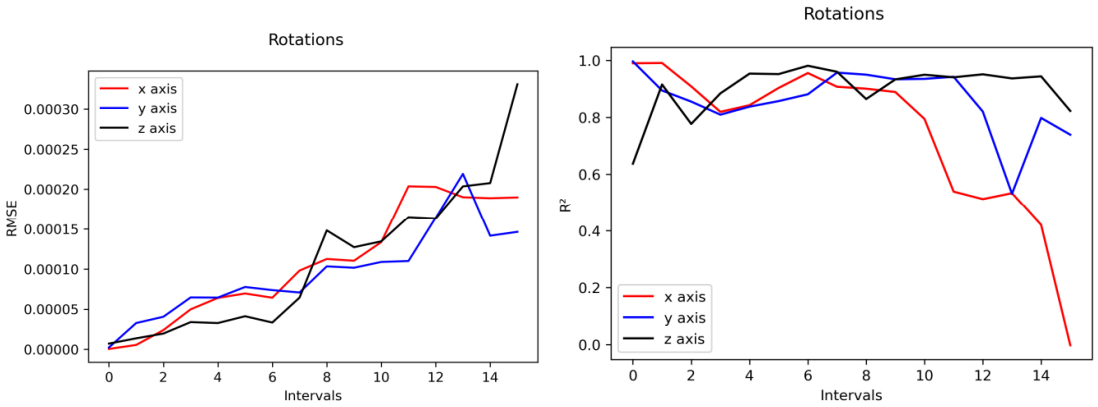

Figure 12: RMSE and R^2 values for each step in Gradient Boost model for rotations

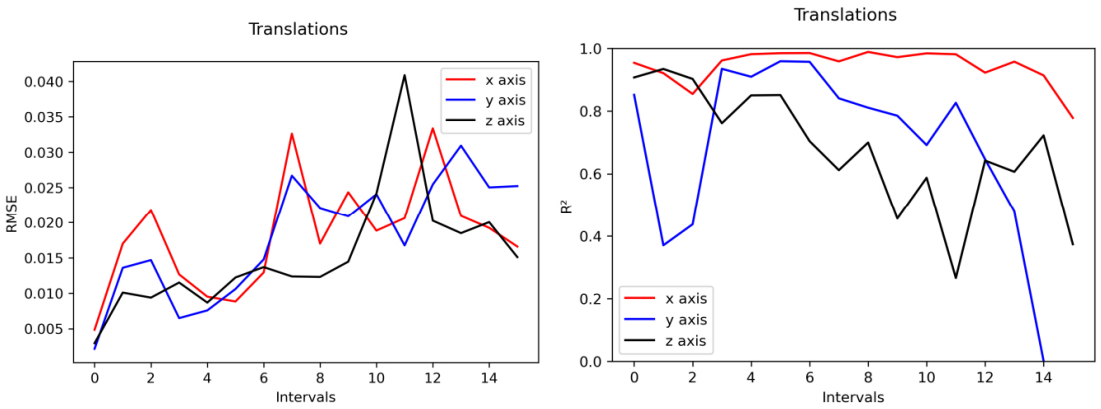

Figure 13: RMSE and R^2 values for each step in Random Forest model for translations

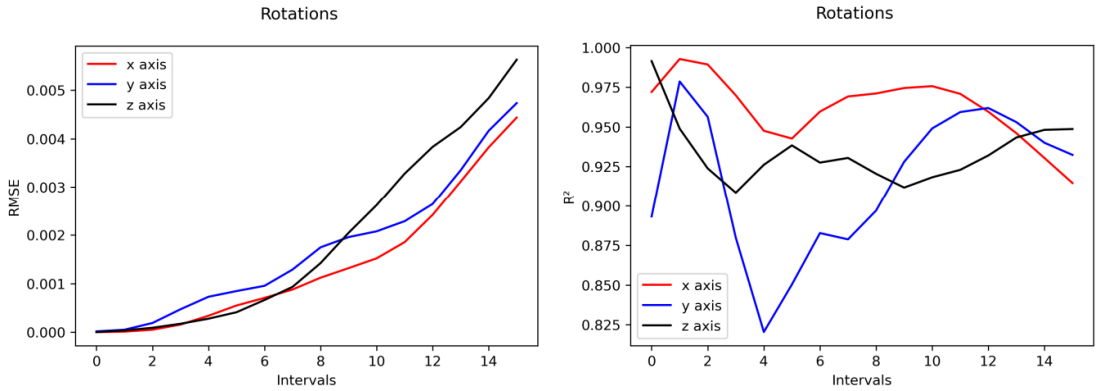

Figure 14: RMSE and R^2 values for each step in Random Forest model for rotations

4. Conclusions

The first and most important conclusion that has been drawn during the development of this project is that there is no perfect algorithm. Each one has its own virtues and defects. The dataset used, its structure and the error metrics used for the evaluation are also key and will affect largely the final result and conclusions.

Looking at the final obtained dataset, some clustering can be seen. This will lead to reduced accuracy of the predictions, especially for Crash Configurations with large differences compared to the ones in the dataset. This clustering can be explained by the fact that most of the simulations are performed using a standardized test configuration. Introducing additional simulations, or even creating them specifically to fill those *gaps* could be beneficial for the model. In these terms, completing the model with information from lateral crashes could help, in particular for simulations with HCPA values close to the frontal/lateral established limit. Introducing information from other manufacturers could also lead to a more complete model, but would require introducing an additional parameter for manufacturer description.

Regarding the results of the project, and considering the error scores obtained, it can be stated that the best fitting algorithms are Support Vector Machine Regression for the translations and Random Forest Regression for the rotations. It can also be seen that the overall performance of Decision Tree-based algorithms is significantly better for rotations compared to the Kernel-based algorithms.

References

[1] Gerald Joy Sequeira. “Evaluation and characterization of crash-pulses for head-on collisions with varying overlap scenarios”. In: (2020).

[2] Patil Kantilal. “Crash Pulse Characterization to Minimize Occupant Injuries in Offset Frontal Crash”. In: (2017).

[3] Zhiqing Cheng. “Optimal Crash Pulse for Minimization of Peak Occupant Deceleration in Frontal Impact”. In: (2005).

[4] Wesley Grimes. “The Effect of Crash Pulse Shape on Occupant Simulations”. In: (2000).

[5] Linus Wågström. “Integrated Safety: Establishing Links for a Comprehensive Virtual Tool Chain”. In: (2019).

[6] Alexandros Leledakis. “A method for predicting crash configurations using counterfactual simulations and real-world data”. In: (2020).

[7] Alexandros Leledakis. “Pre-Crash and In-Crash Car Occupant Safety Assessment”. PhD thesis. Chalmers University of Technology, 2021.

CRASH PULSE PREDICTION APPLIED TO FRONTAL CRASH CONFIGURATIONS

Autor: Mira Arana, Pablo
Director: López Valdés, Francisco José
Entidad Colaboradora: Volvo Cars AB

RESUMEN DEL PROYECTO

1. Introducción

La seguridad de los ocupantes de un vehículo es uno de los principales retos a la hora de diseñar un vehículo. Dado que el número de vehículos aumenta cada año en todo el mundo, mejorar su rendimiento en caso de choque en lo que respecta a la protección de los ocupantes es clave para reducir el número de lesiones y muertes. En este sentido, el estudio de lo que ocurre durante un choque es crucial para evaluar y mejorar la seguridad. Para ello, se realizan numerosas pruebas de choque físicas y virtuales, probando diferentes escenarios y configuraciones. Ambas metodologías tienen sus limitaciones, siendo las principales el consumo de tiempo y los elevados costes.

La introducción del Machine Learning en este campo puede aportar numerosos beneficios, aprovechando la existencia de años de pruebas realizadas y registradas. El objetivo principal de este proyecto es ayudar a la evaluación de la respuesta motriz que tiene el vehículo durante el choque. Esta respuesta se denomina Pulso de Choque, y su importancia y efecto en la gravedad del choque ha sido ampliamente estudiada y probada.[1][2][3]

El comportamiento global del vehículo durante una colisión puede variar en gran medida en función de cómo se produzca el choque. Se pueden establecer muchos tipos de choques, pero el principal es la división entre choques laterales y frontales. Mientras que en el primero tienen gran importancia las intrusiones en el habitáculo y las aceleraciones laterales, el segundo viene determinado en gran medida por las deceleraciones sufridas. Este proyecto se centrará en las colisiones frontales.

El objetivo de este proyecto es desarrollar un algoritmo de Inteligencia Artificial que sea capaz de predecir el Pulso de Choque resultante con una Configuración de Choque dada. La definición de Configuración de Choque que se utilizará está definida en [4] y fue posteriormente utilizada en [5].

2. Metodología

El proyecto consta de dos etapas principales. La primera es el proceso de análisis de datos, donde se registra toda la información de las pruebas anteriores, se filtra y se preprocesa para introducirla en el algoritmo de IA. La segunda es el diseño de este algoritmo de IA, su entrenamiento, testing y evaluación.

2.1 Análisis de Datos

El proceso de análisis de datos se realiza para crear un conjunto de datos válido que pueda ser entendido por el modelo de IA. El proceso para hacerlo se describe en la Figura 1.

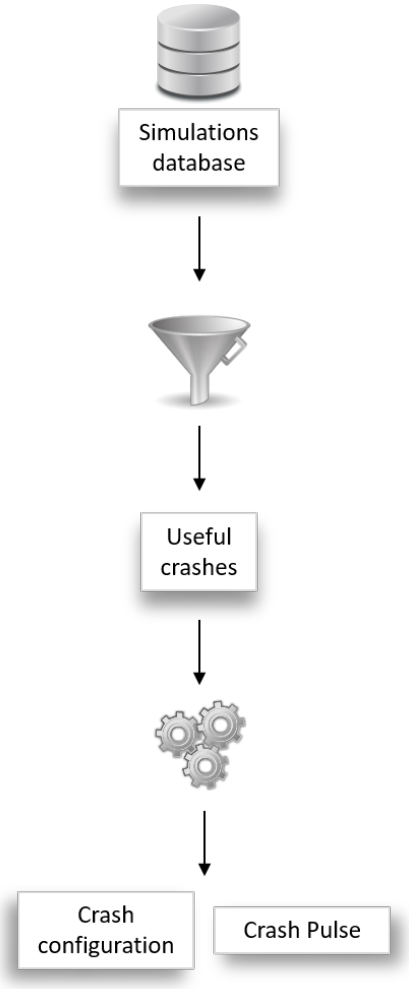

Figura 1: Proceso de Recogida de Datos.

- **Selección**. De toda la información disponible, se deben seleccionar las simulaciones útiles y pertinentes para este proyecto. Esto incluye simulaciones de coche contra coche y de coche contra barrera móvil, con una longitud mínima de pulso de choque registrado y dentro de los límites establecidos para los choques frontales y traseros.

- **Preprocesado**. Los datos necesarios para la aplicación de la IA deben extraerse de las simulaciones seleccionadas. Esta extracción de información puede dividirse en dos partes diferenciadas:

  - *Configuración de Choque*. Esta es el input para el modelo de IA. Los parámetros que describen el choque deben calcularse a partir de la información bruta de la simulación. La información de los archivos **d3plot** y **nodout** se ha utilizado para detectar cada objeto que colisiona mediante un proceso de clustering, calcular su velocidad (HS y OS), sus masas (HM y OM) y los ángulos que describen su posición relativa. Estos ángulos son el ángulo del punto de colisión del vehículo anfitrión (HCPA), el ángulo del punto de colisión del vehículo oponente (OCPA) y el ángulo de guiñada del oponente (OYA). Los dos primeros están referidos al primer punto de contacto (FPOC). Para poder comparar entre configuraciones con diferencias dimensionales, hay que realizar una normalización, reduciendo la forma del vehículo a un cuadrado unitario.
    Una descripción gráfica está representada en las figuras 2 y 3, y un ejemplo de la configuración de choque resultante se representa en la tabla 1.
    El límite establecido para las colisiones frontales/traseras es [-45,45] y [135,225] en lo que respecta al HCPA.

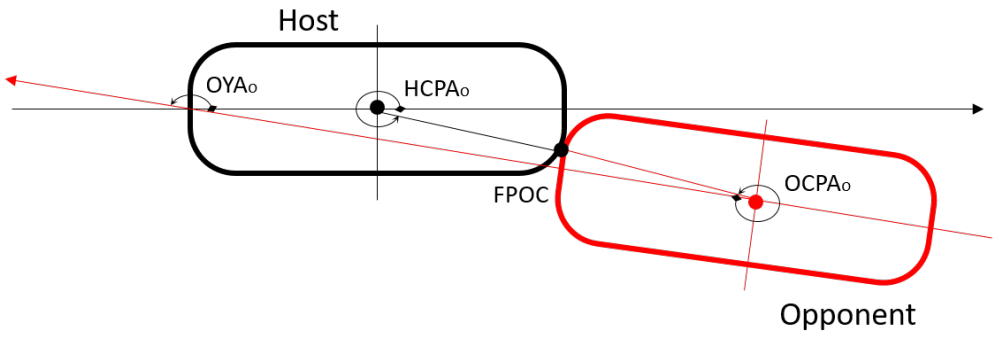

Figura 2: Ángulos de la Configuración de Choque [4]

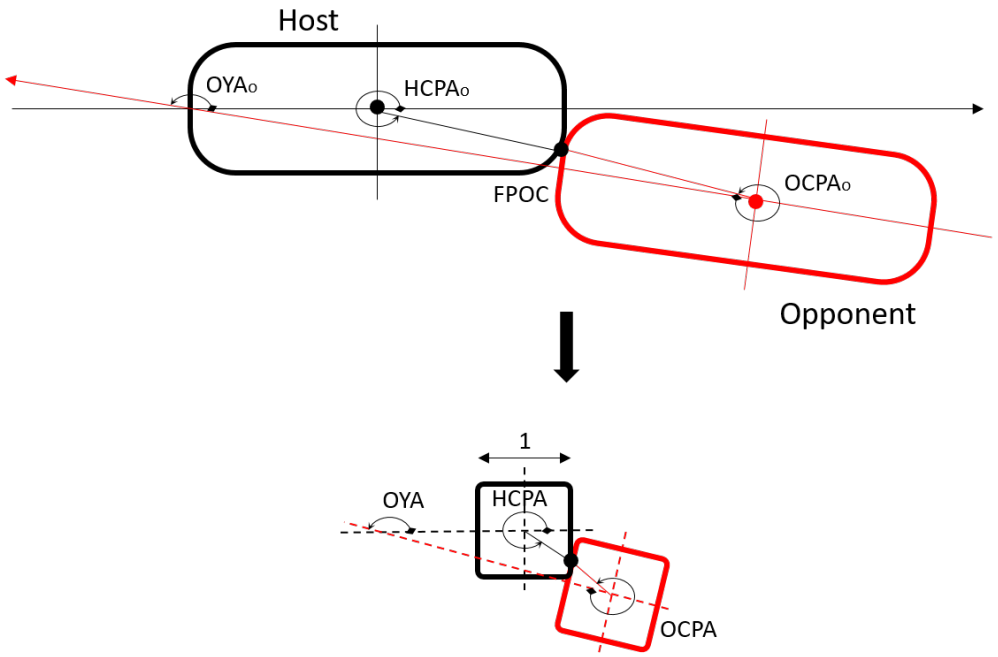

Figura 3: Normalización de la Configuración de Choque [4]

HCPA	-38º
OCPA	-26º
OYA	160º
Host speed	13.5 m/s
Opponent speed	21 m/s
Host mass	2178.9 kg
Opponent mass	2335.4 kg

Cuadro 1: Parámetros de la Configuración de Choque

- *Pulso de choque*. Este es el output del modelo. La información del pulso de choque se extrae utilizando un script preexistente, que calcula el pulso de choque en términos de velocidades. El pulso de choque está formado por 6 series temporales, 3 ejes traslacionales y 3 ejes rotacionales. Para las traslaciones, la información se ha convertido en aceleración. Por su parte, las rotaciones se han convertido en ángulos. Cada serie temporal se ha recortado a 80 ms y se ha descrito utilizando 16 timesteps. La información resultante que describe el pulso de choque de cada simulación está formada por 96 puntos unidimensionales.

2.1 Modelado de la IA

Los datos se han dividido en dos conjuntos de datos diferentes. El primero se introducirá en el modelo con información de input y output, para que pueda aprender el output esperado. Este conjunto se denomina conjunto de entrenamiento. El segundo se introducirá sólo con información del input, para que el modelo haga predicciones a ciegas que luego podrá comparar con el output real. Este conjunto se denomina conjunto de test.

Como el parámetro de salida es un valor continuo, el modelo de predicción será de regresión. Se han probado cuatro algoritmos diferentes. Estos cuatro modelos son Neural Networks, Support Vector Machine Regression, Gradient Boost Regression y Random Forest Regression. Los dos primeros algoritmos están basados en un Kernel, mientras que los dos últimos están basados en Árboles de Decisión. Cada algoritmo ha sido modelado individualmente para obtener las mejores predicciones posibles.

3. Resultados

El conjunto de datos resultante está formado por 284 puntos, 232 (81,7 % del conjunto de datos) para el conjunto de entrenamiento y 52 (18,3 % del conjunto de datos) para el conjunto de test. Esta división se representa en la figura 4, donde el conjunto de datos queda representado en términos de HCPA (eje x) y OCPA (eje y).

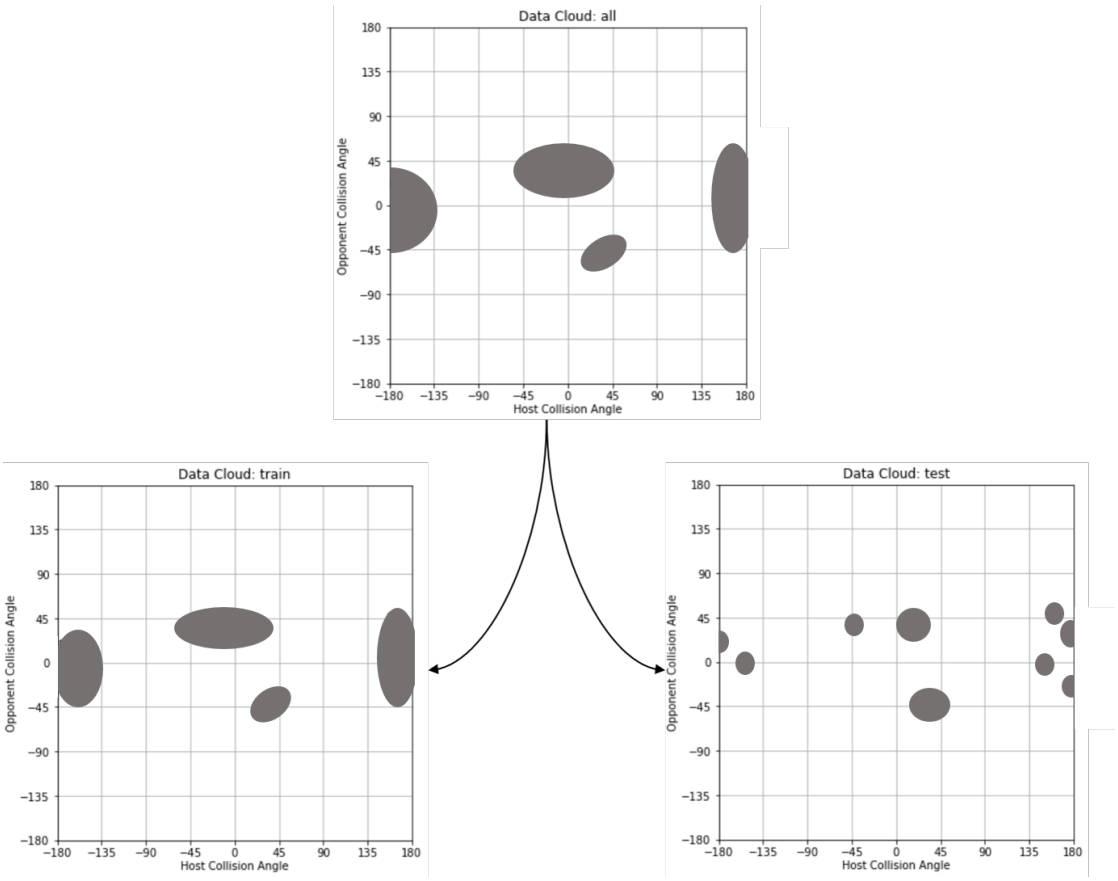

Figura 4: Nube de datos y división

Los cuatro modelos se han probado utilizando las métricas R^2 y RMSE, tanto para cada timestep/predicción como para la estructura completa de traslaciones/-rotaciones. Las puntuaciones medias se representan en la tabla 2, y en las figuras 5, 6.

	Traslaciones		Rotaciones	
Algoritmo	RMSE	R^2	RMSE	R^2
Neural Network	0.020	0.728	0.0024	0.103
Support Vector Machine	0.017	0.805	0.0023	0.319
Gradient Boost	0.019	0.747	0.000126	0.834
Random Forest	0.0189	0.751	0.00229	0.937

Cuadro 2: Métricas de error para cada modelo

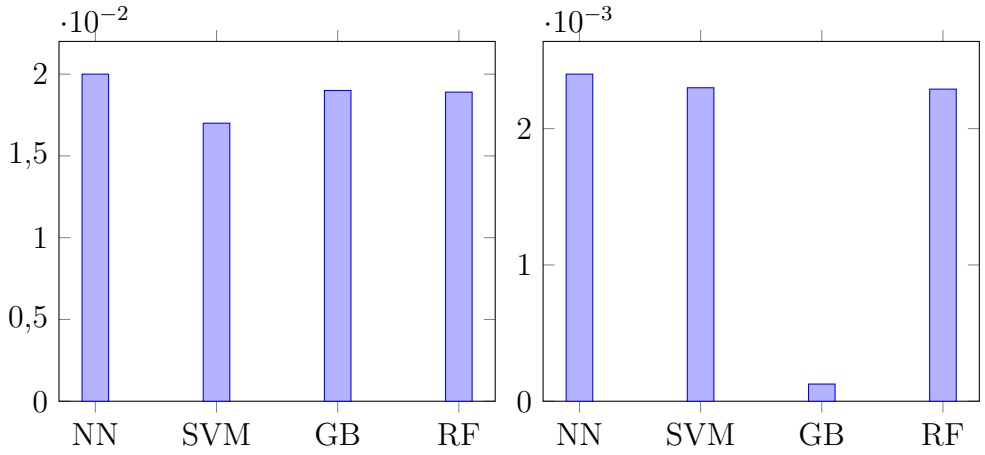

Figura 5: Comparativa de valores de RMSE para traslaciones (derecha) y rotaciones (izquierda)

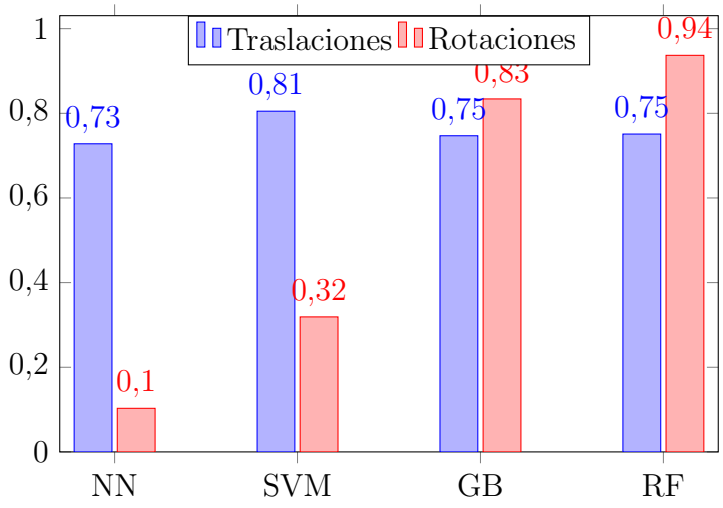

Figura 6: Comparativa de R^2

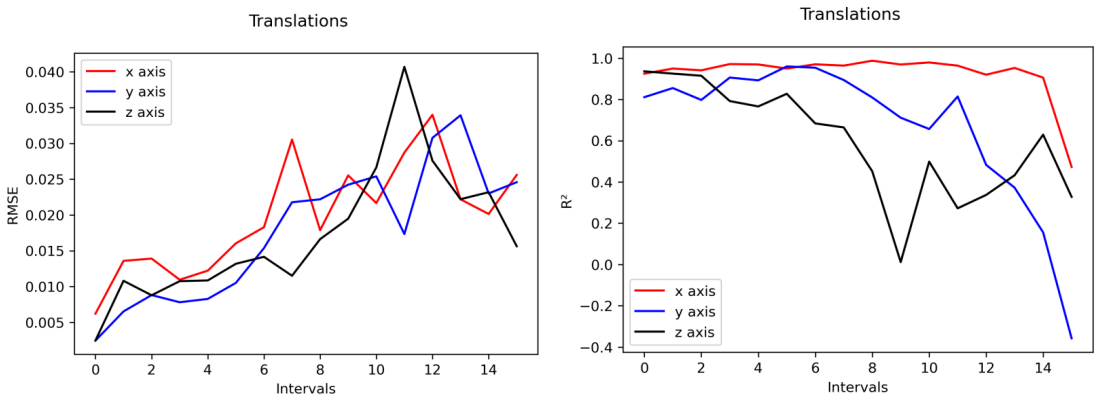

Figura 7: Valores de RMSE y R^2 para cada timestep en traslaciones del modelo de Neural Network

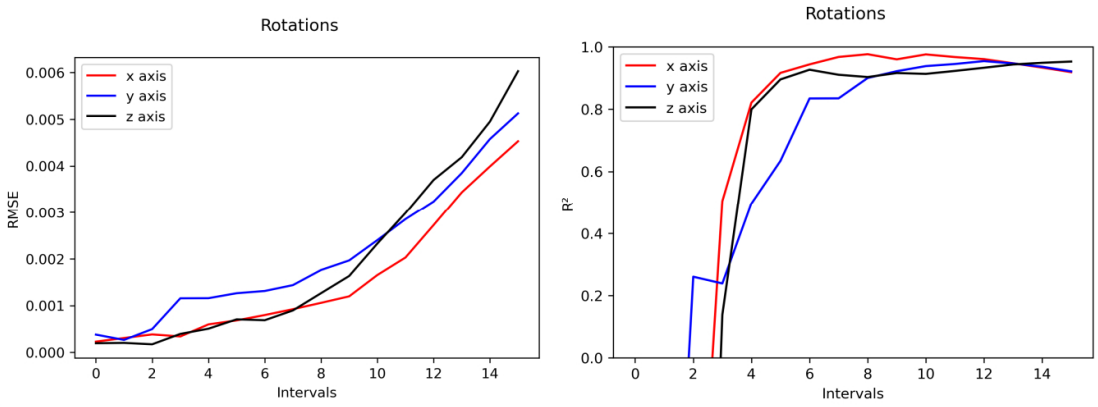

Figura 8: Valores de RMSE y R^2 para cada timestep en rotaciones del modelo de Neural Network

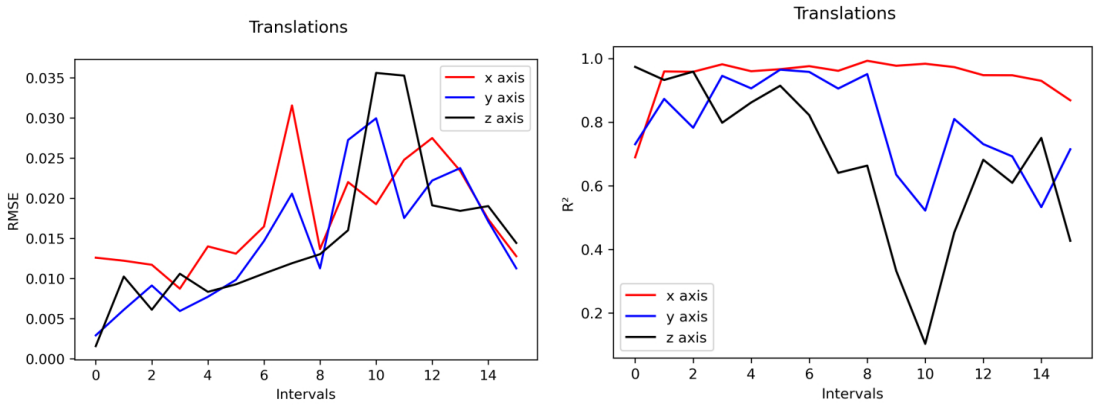

Figura 9: Valores de RMSE y R^2 para cada timestep en traslaciones del modelo de Support Vector Machine

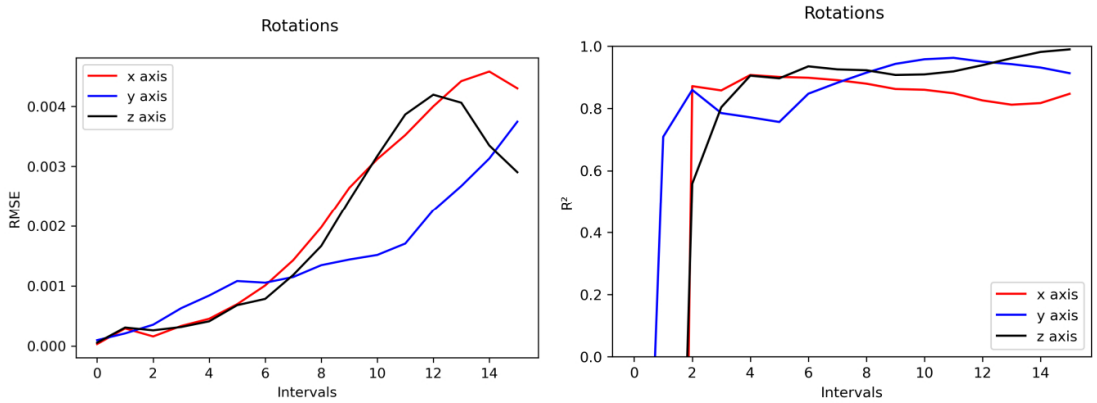

Figura 10: Valores de RMSE y R^2 para cada timestep en rotaciones del modelo de Support Vector Machine

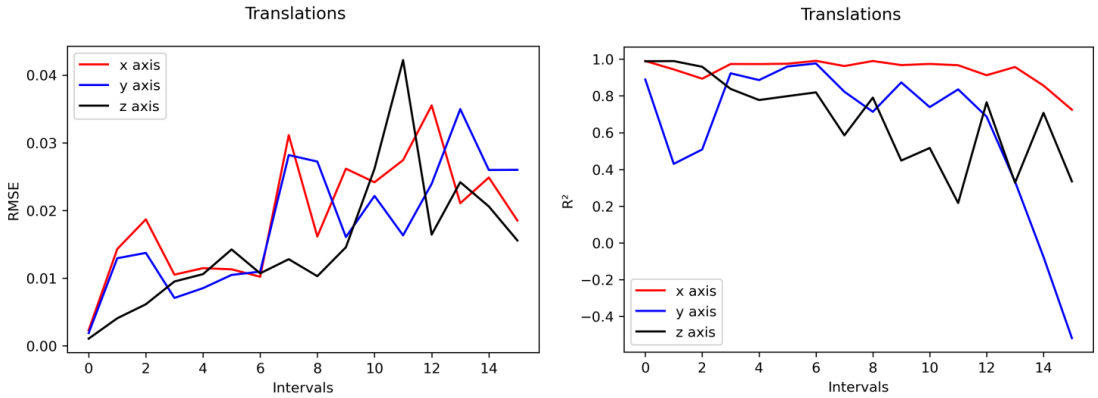

Figura 11: Valores de RMSE y R^2 para cada timestep en traslaciones del modelo de Gradient Boost

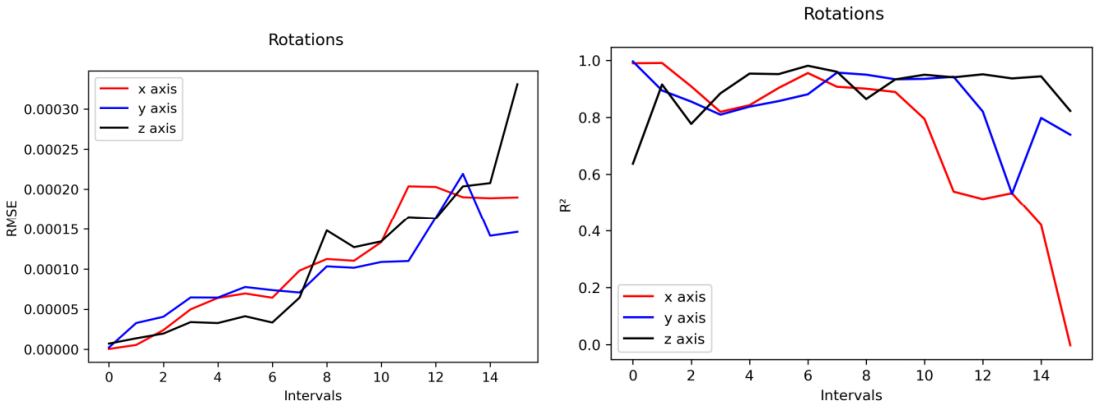

Figura 12: Valores de RMSE y R^2 para cada timestep en rotaciones del modelo de Gradient Boost

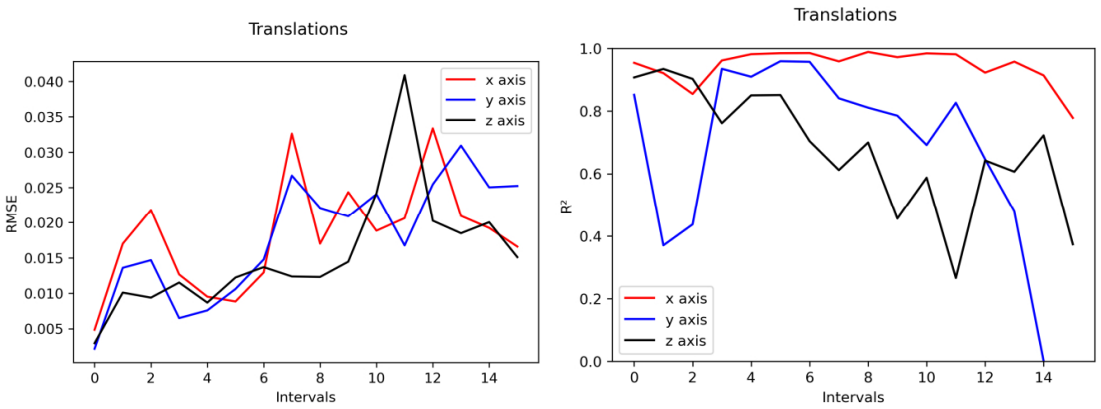

Figura 13: Valores de RMSE y R^2 para cada timestep en traslaciones del modelo de Random Forest

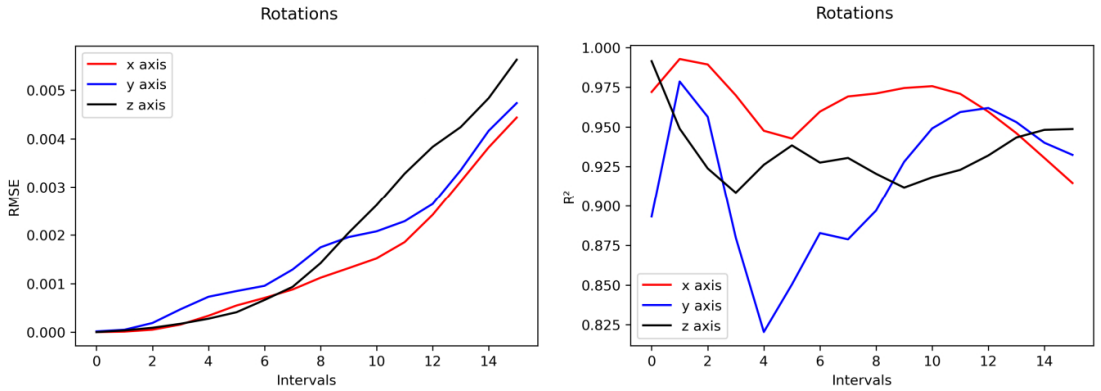

Figura 14: Valores de RMSE y R^2 para cada timestep en rotaciones del modelo de Random Forest

4. Conclusiones

La primera conclusión que se ha extraído durante el desarrollo de este proyecto es que no existe un algoritmo perfecto. Cada uno tiene sus propias virtudes y defectos. El conjunto de datos utilizado, su estructura y las métricas de error utilizadas para la evaluación también son clave, y afectarán en gran medida al resultado final y a las conclusiones.

Si se observa el conjunto de datos obtenido, se puede apreciar cierto clustering. Esto conducirá a una menor precisión de las predicciones, especialmente para las configuraciones de choque con grandes diferencias en comparación con las del conjunto de datos. La introducción de simulaciones adicionales, o incluso la creación de las mismas específicamente para rellenar esas *lagunas*, podría ser beneficiosa para el modelo. En este sentido, completar el modelo con información de colisiones laterales podría ayudar, en particular para las simulaciones con valores de HC-PA cercanos al límite frontal/lateral establecido. Introducir información de otros fabricantes también podría conducir a un modelo más completo, pero requeriría introducir un parámetro adicional para la descripción del fabricante.

En cuanto a los resultados del proyecto, y teniendo en cuenta las puntuaciones de error obtenidas, se puede afirmar que los algoritmos que mejor se ajustan son Support Vector Machine para las traslaciones, y Random Forest para las rotaciones. También se puede observar que el rendimiento global de los algoritmos basados en árboles de decisión es significativamente mejor para las rotaciones en comparación con los algoritmos basados en kernels.

Referencias

[1] Gerald Joy Sequeira. "Evaluation and characterization of crash-pulses for head-on collisions with varying overlap scenarios". En: (2020).

[2] Patil Kantilal. "Crash Pulse Characterization to Minimize Occupant Injuries in Offset Frontal Crash". En: (2017).

[3] Zhiqing Cheng. "Optimal Crash Pulse for Minimization of Peak Occupant Deceleration in Frontal Impact". En: (2005).

[4] Linus Wågström. "Integrated Safety: Establishing Links for a Comprehensive Virtual Tool Chain". En: (2019).

[5] Alexandros Leledakis. "A method for predicting crash configurations using counterfactual simulations and real-world data". En: (2020).

[6] Alexandros Leledakis. "Pre-Crash and In-Crash Car Occupant Safety Assessment". Tesis doct. Chalmers University of Technology, 2021.

Master's Degree in Industrial Engineering

Master's final project

Crash Pulse Prediction applied to Frontal Crash Configurations

Author
Pablo Mira Arana

Supervised by
Francisco José López Valdés

Madrid
June 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Road safety is one of the biggest concerns of modern society. In 2016, 1.35 million road traffic deaths were estimated worldwide [1], and the number of injuries is considered to be over 10 times the number of deaths.

As the number of vehicles increases every year, making these vehicles safer is crucial for limiting injuries and deaths. Vehicle safety can be evaluated using 2 different procedures. The first one, and the oldest one, is to perform physical tests. These tests require a large financial investment to have the right equipment and the right facilities. Apart from that, each test means a high cost and months of preparation. This is one of the reasons why the second procedure was developed. This procedure is to simulate these tests with FE simulations. By doing so, the cost of equipment and facilities is drastically reduced. But these simulations also have their flaws. They are also expensive computational and time-wise.

Using both physical tests and simulations, occupant safety in vehicles has been improved over the last decades, but with the emergence of AD and ADAS, a paradigm shift has occurred. Therefore, new scenarios may need to be evaluated, and new variables may need to be considered. [2][3]

To help the evaluation of this new paradigm, Data Science and AI could have an important role in vehicle safety.

1.1 Vehicle safety evaluation

Nowadays, a large number of tests are done via simulations. Many reasons are explaining this. Cost reduction, the possibility of recreating scenarios that are not ethically right to test physically, etc. The complete process of occupant safety evaluation consists of two main phases: pre-crash and in-crash. The pre-crash phase has gained a lot of importance with the growth of ADAS, as these systems are designed to try to avoid the crash, or mitigate it. Therefore, the resultant crash might be varied with the action of ADAS. In this first phase, the resulting crash is determined.

When the final crash configuration (how is the crash occurring) is defined, the in-crash phase is developed. In this in-crash phase, the crashworthiness (how the structure of the vehicle behaves) is evaluated, to then evaluate the response of the occupant (using an ATD (Anthropomorphic Test Device) or HBM (Human Body Model)) and of the restraint systems and safety features.

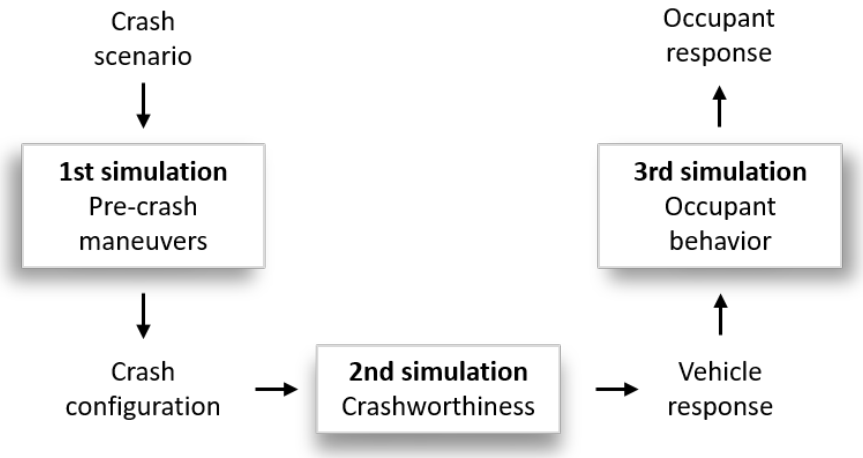

Figure 1.1: Simulation flow example.

Considering the flow chart represented in Figure 1.1 as a possible simulation flow, this particular project is focused on predicting one of the outputs of the 2nd simulation, the crash pulse, from a given crash configuration. This crash pulse will define to a large extent the occupant motions, defining too the crash severity [4]. Therefore, the crashworthiness evaluation is essential when trying to reduce this severity. The predicted crash pulse can be used, for example, to evaluate the pre-crash phase regarding possible resulting crash configurations.

1.1.1 Frontal crashes

Frontal crashes include both Head-on and Rear-end collisions, as both involve the frontal/back end of the vehicle. These two types of collisions have significant differences. In rear-end collisions, as represented in Figure 1.2, both vehicles are heading in the same direction. On the other hand, in head-on collisions, as represented in Figure 1.3, the vehicles colliding have opposite directions. Therefore, the characteristics of the crash regarding occupant response and suffered accelerations/impacts will differ largely.

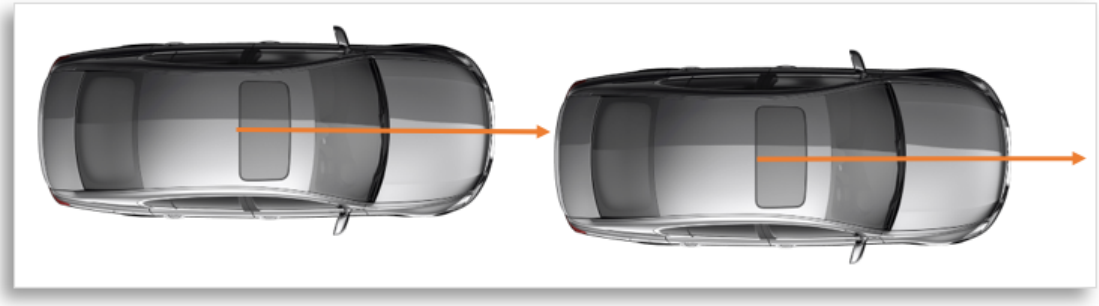

Figure 1.2: Rear-end collision.

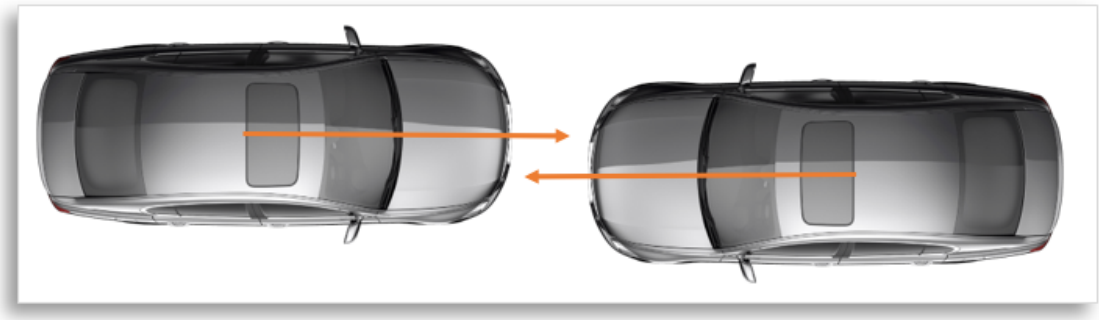

Figure 1.3: Head-on collision.

1.2 Keywords

Machine Learning, Artificial Intelligence, Crash Configuration, Crash Pulse, Time Series, Python, Prediction, Regression, Frontal Crashes, Simulations

1.2.1 Abbreviations

- ML → Machine Learning
- DL → Deep Learning
- AI → Artificial Intelligence
- CC → Crash Configuration
- CP → Crash Pulse
- c2c → Car to car
- c2b → Car to barrier
- AD → Autonomous Driving
- ADAS → Advanced Driver Assistant Systems
- CoG → Center of Gravity
- FPOC → First Point of Contact
- RF → Random Forest
- NN → Neural Network
- SVM → Support Vector Machine
- GB → Gradient Boost

Chapter 2

Motivation

This project was born from the need of processing large sets of data efficiently, investigating numerical methods for predicting the acceleration pulse of a vehicle impact with a given crash configuration.

Vehicle safety needs constant research and development to adapt to evolving mobility. With the growth of ADAS and the research regarding AD, safety may experience a major change in its objectives and can result in a bigger need for it to evolve.

With this pulse prediction, the study of different crash scenarios can be faster and more cost-efficient compared to virtual simulations and crash tests. This can help in the occupant safety enhancement, being useful for assessing the effect of pre-crash maneuvers on the severity of the crash, evaluating the occupant response, etc. This project can also be a first step for the inclusion of Artificial Intelligence in vehicle safety evaluation.

Alongside the development of the project, extensive knowledge of Data Analysis and Machine Learning methodologies will be acquired. These two fields are becoming increasingly important, since in every industry the volume of data is increasing exponentially, and these methodologies are needed to process it and obtain useful information from it.

Chapter 3

Objectives

This project aims to develop a method that predicts the resulting crash pulse of a vehicle impact from a given crash configuration, using ideally both simulation and test data. To define the project scope, 4 main objectives have been identified:

1. **Explore existing methods.** Search for similar projects, and developed methodologies suitable for the project.

2. **Crash configuration method.** Create a method that automatically parses through test and simulation data and extracts the desired crash configuration parameters. These crash configurations shall be valid and shall have a corresponding crash pulse, as it is needed to first train, then test the algorithm.

3. **Crash pulse forecasting.** Create and train a method that predicts crash pulses with a given standardized crash configuration. This method will read the crash configuration parameters and generate a 3-axis translation and 3-axis rotation time-series.

4. **Accuracy evaluation.** Evaluate the accuracy of predicted crash pulses in comparison with original crash pulses. To do so, the data set of crashes will be split into train and test data sets.

Chapter 4

State of the art

In this chapter, an in-depth explanation of a variety of topics will be carried out, in order to give the reader a better understanding of the knowledge domains and specific tools that have been used during the development of this project.

4.1 Crash Configuration

The Crash Configuration is a set of parameters that define the principal characteristics of how a crash occurs. It is referred to as the ego vehicle, also known as the Host vehicle. The Crash Configuration definition that will be used for this project is the Volvo Parametric Crash Configuration (VPARCC) [5], which includes the parameters description and the Crash Configuration normalization.

This Crash Configuration relies on the identification of the First Point of Contact (FPOC) for the parameter definition. As it uses a top view of the crash, the configuration is described in a two-dimensional plane.

To properly define this Crash Configuration, the boundaries of both colliding objects must be known, as well as the geometrical center points of both objects, their velocities, and directions. With this information, the positional angles and the velocities that describe the Crash Configuration can be defined. For this project, an object description parameter has been added, with mass being the chosen one. The resulting parameters are:

1. **Host Collision Point Angle (HCPA)**. This angle is defined as the counter-clockwise angle between the host center plane and the FPOC.
2. **Opponent Collision Point Angle (OCPA)**. This angle has a similar

definition as the HCPA, but the reference is taken from the opponent vehicle/object, considering its center plane.

3. **Opponent Yaw Angle (OYA)**. This angle is defined as the counterclockwise angle between the direction of the host vehicle and the direction of the opponent vehicle.
4. **Host Speed (HS)**. The velocity magnitude of the Host vehicle.
5. **Opponent Speed (OS)**. The velocity magnitude of the Opponent vehicle.
6. **Host Mass (HM)**. The mass of the Host vehicle, considering possible occupants.
7. **Opponent Mass (OM)**. The mass of the Opponent vehicle, considering possible occupants.

The three positional angles are represented in Figure 4.1 for a better understanding.

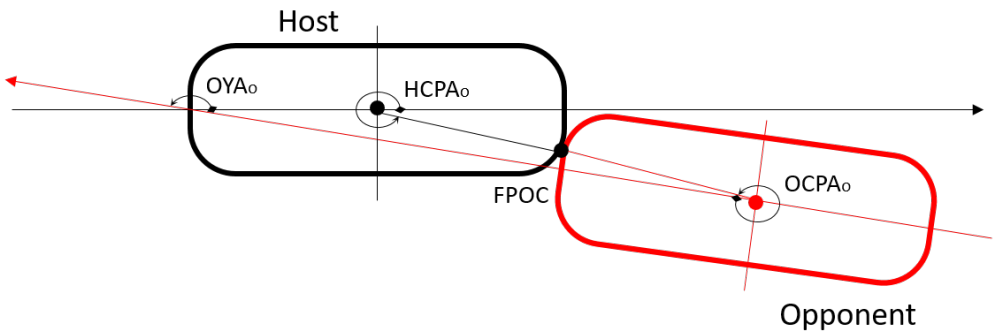

Figure 4.1: Crash Configuration Angles [5]

This Crash Configuration is dependant of the dimensions of the colliding vehicles. To compare Crash Configurations from different vehicles, this dependency must be eliminated. To do so, a normalization of the vehicle is performed, converting it to a square unit. This normalization will allow not only to compare Crash Configurations from vehicles with different dimensions, but to have a better visualization tool for large sets of Crash Configurations.

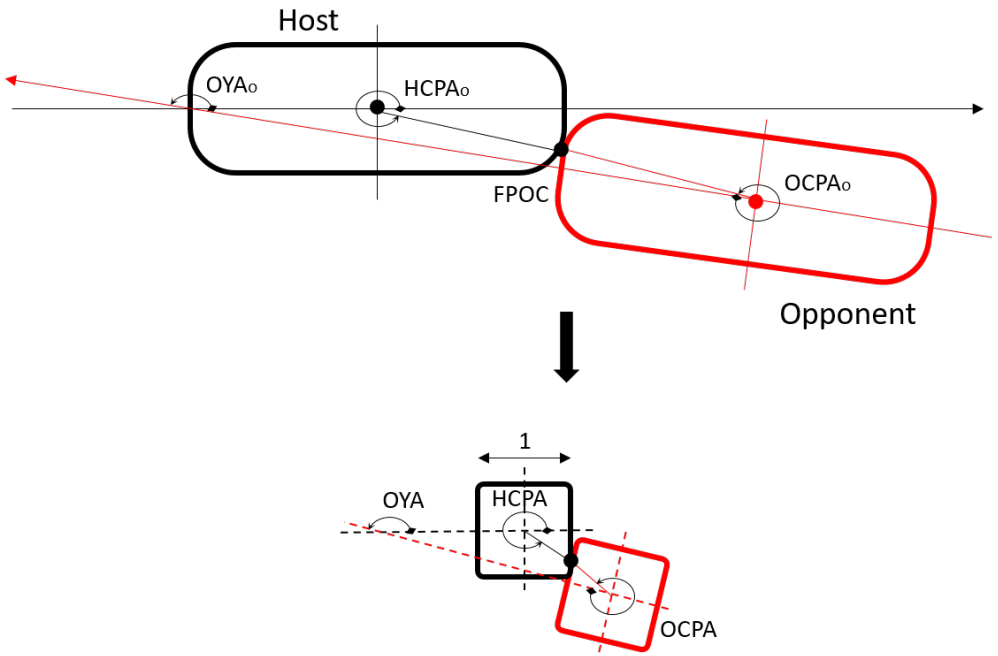

Figure 4.2: Crash Configuration Normalization [5]

The Crash Configuration and its normalization are further explained in [5] and [6].

4.1.1 Frontal and Rear-end Crashes

As seen in Section 1.1.1, Rear-end and Head-on collisions represent a large percentage of the c2c collisions. However, the limits that define the difference between a frontal/rear-end collision and a lateral collision are not defined. For the purpose of this project, this limit has been set at 45^{o} for the normalized HCPA, regarding both corners in the anterior and posterior parts. This is represented in Figures 4.3 and 4.4.

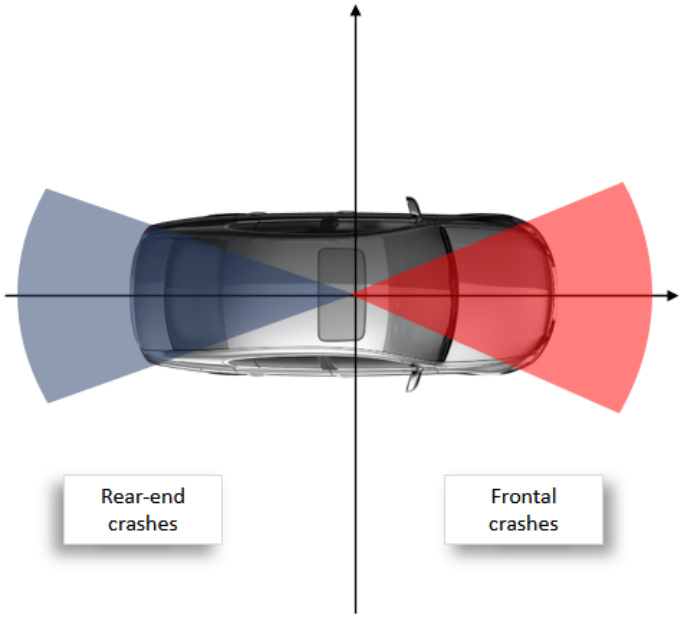

Figure 4.3: Frontal Crashes Discrimination

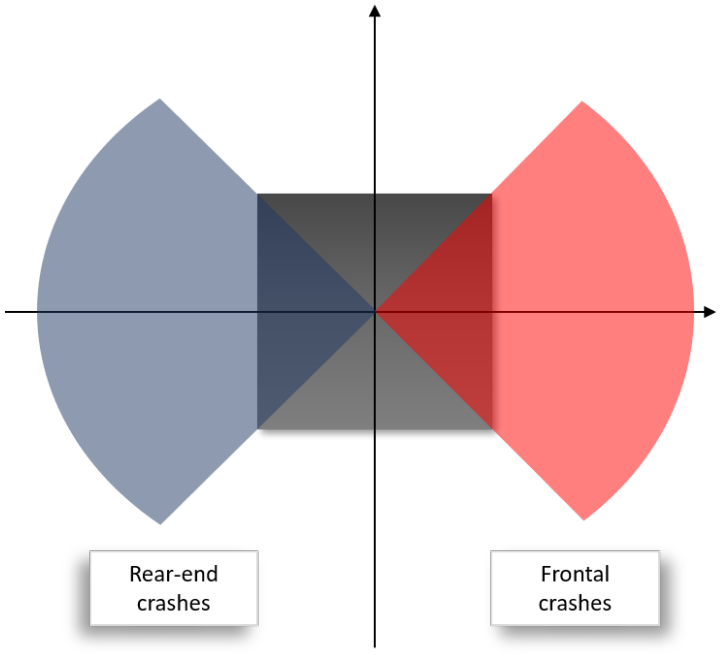

Figure 4.4: Frontal & Rear-end crashes discrimination

4.2 Crash Pulse

A vehicle crash is a very complex event, where hundreds of parameters influence its severity. In order to enhance occupant safety, one of the most important outputs of a vehicle crash is the Crash Pulse, which is the movement history of the vehicle during the crash. It can be represented using displacements, velocities, or accelerations, being this last one the most common and the most straightforward when studying its influence in the occupant response. Its shape, peak values, and duration provide important information over how the occupant response will be during the in-crash phase. In turn, these Crash Pulse parameters are very dependent on the initial Crash Configuration.

Numerous studies have been carried out in order to minimize occupant injuries, especially regarding frontal crashes. Finding the Crash Pulse parameters that define the severity of a crash has become a key factor when studying vehicle safety. See *Optimal Crash Pulse for Minimization of Peak Occupant Deceleration in Frontal Impact* [7], *The Effect of Crash Pulse Shape on Occupant Simulations* [8], and *Crash Pulse Characterization to Minimize Occupant Injuries in Offset Frontal Crash* [9] for further information about this topic.

The Crash Pulse can be represented in different ways. For the scope of this project, it will be described using 6 time series, being 3-axis translations and 3-axis rotations. This is represented in Figure 4.5.

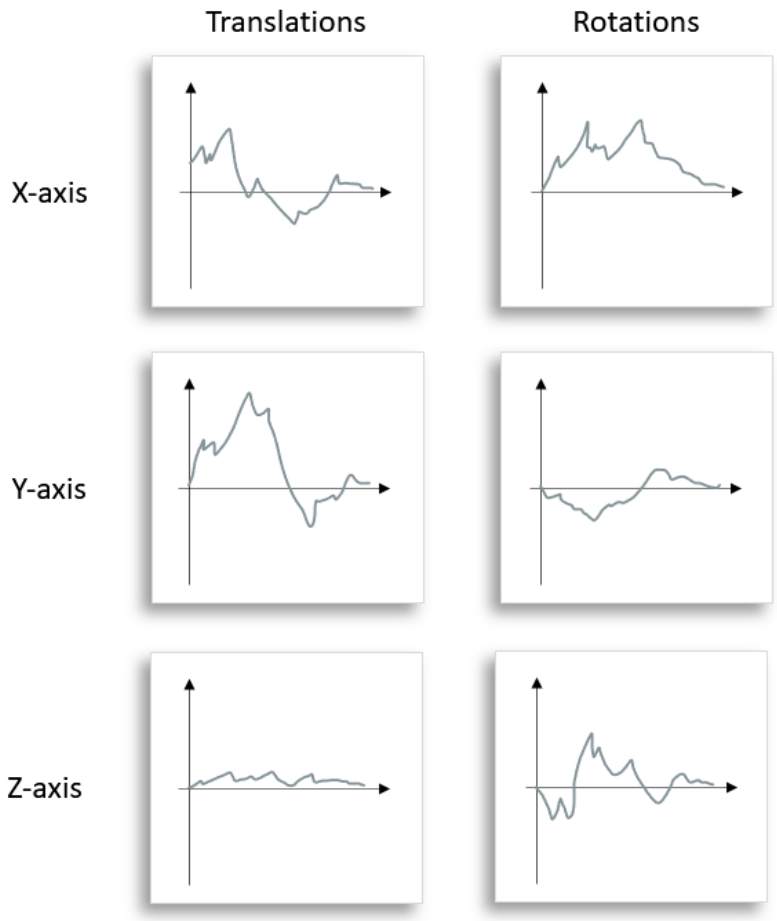

Figure 4.5: Crash Pulse example

4.3 Machine Learning

4.3.1 K-Means Clustering

Clustering is a process in which the objective is to group a set of data points/objects in different groups, where all the objects in the same group have more similar characteristics between them than with the objects in other groups. Its commonly used in data mining applications and statistical data analysis.

Clustering is an Unsupervised Learning methodology. This means that there is not a specific target to predict, and the data provided is unlabeled. The algorithm finds hidden patterns in data on its own, with given characteristics of the data (also called dimensions).

K-Means clustering is one of the most popular ML algorithms. This is due to its versatility and simplicity. It is a partitional clustering technique, based on centroid identification. The k parameter refers to the number of clusters that must be identified, and it a user-set parameter. K-Means' objective is to reduce the sum of squares in each cluster, considering the mean value in each cluster. This process is done by finding the centroid (arithmetic mean of all the points belonging to the cluster) of each cluster and calculating the distance of each point of the cluster to this centroid. This is an iterative process, searching for convergence of the centroid positions. It starts with randomly selected centroids, iterating until the centroids are stabilized or a defined number of iterations have been achieved.

In clustering processes, it is important to select correctly the dimensions that it may take into account, as well as giving them an adequate weight. If one of the dimensions has a magnitude ten times larger than the others, the algorithm will give this dimension more weight when finding the clusters.

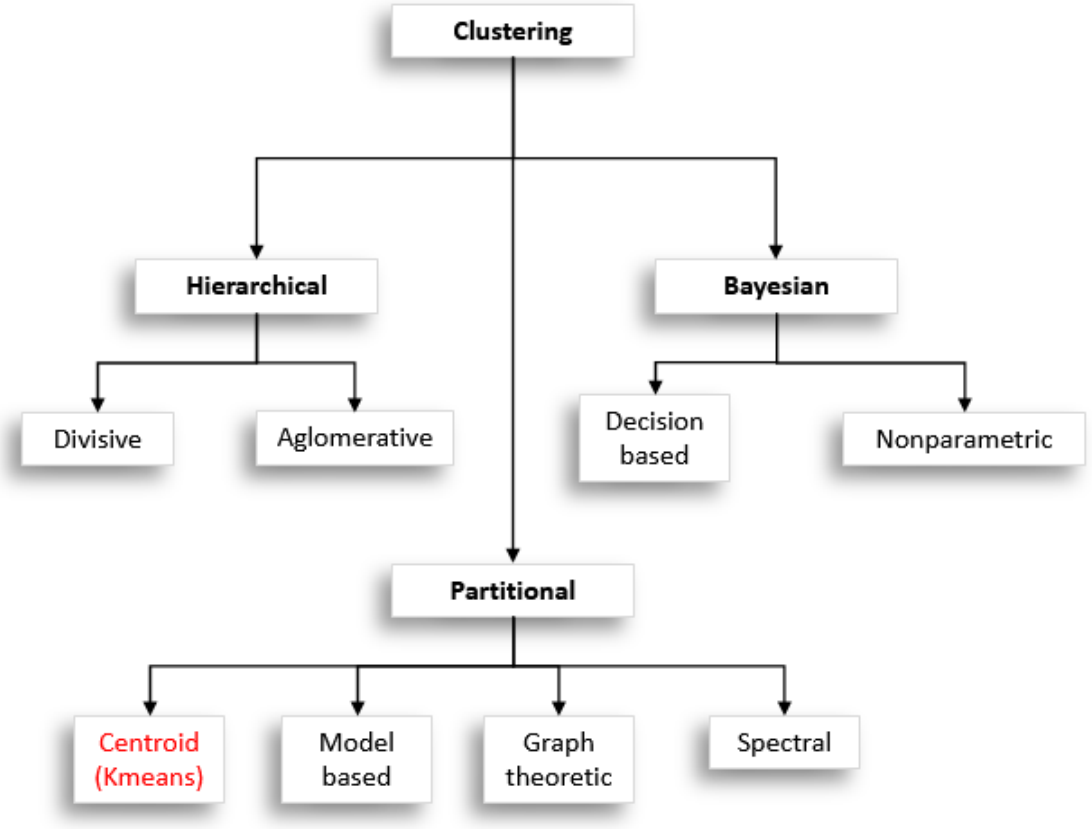

Figure 4.6: Clustering techniques

In Python, the scikit-learn library includes a K-Means clustering class [10]. This class allows easy implementation a clustering process with few preparation and defined parameters. Defining the number of desired clusters and giving the data set with the important dimensions, the process will be performed automatically, returning an object with all the cluster information. This object includes the centroids coordinates, the labeled data points, etc.

4.3.2 Regression

In contrast with the Clustering methodology described before, the Regression methodology is a Supervised Learning methodology. This means that it uses labeled datasets, enabling the model to measure its accuracy and learn over time.

Regression is a learning method that considers relationships between a dependent variable (the variable that the model tries to predict) and independent variables and is widely used to predict numerical values. The general expression of a regression model is shown in equation 4.1, where x_i represent the independent variables, and β_i represent the coefficients of these variables, called parameters.

$$Y = f(x_i, \beta_i) \tag{4.1}$$

There are numerous regression models, being divided primarily into linear and non-linear regression. In linear regression, the dependent variable Y is a linear combination of the parameters. Therefore, even with quadratic expressions for the independent variables, if the parameters β_i are linear, the regression will be linear. If there is only one independent variable, it is called Simple Linear Regression. If multiple independent variables are affecting the prediction of the dependent variable, it is called Multiple Linear Regression.

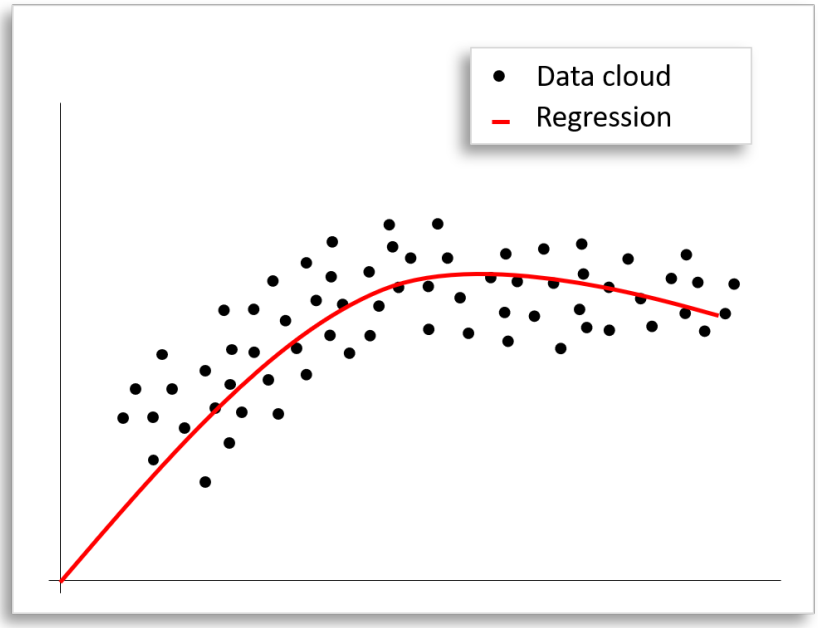

Figure 4.7: Regression example

The difference between linear and non-linear regression is that, unlike linear regression, where all the terms in the model definition are either constants or parameters multiplied by independent variables, non-linear regression models are not constrained to this. Then, its basic definition is represented in equation 4.2. Non-linear regression models include Polynomial Regression, Logistic Regression, etc.

$$Y \sim f(x_i, \beta_i) \tag{4.2}$$

When performing regression, the model's objective is to estimate the values of the parameters β_i to fit the prediction the best way possible. To do so, the difference between the predicted dependent variable and the actual value of this variable must be minimized. This difference is called residual and is usually assessed using sum of squares to avoid negative values. The Residual sum of squares equation is defined in 4.3.

$$RSS = \sum_{i}^{n} (y_i - \hat{y}_i)^2 \tag{4.3}$$

To assess the fitting of the model for the data, the Residual standard error (RSE) and the R^2 functions are commonly used.The functions defining these error metrics are detailed in equations 4.4 and 4.5.

$$RSE = \sqrt{\frac{1}{n-2} RSS} \tag{4.4}$$

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{RSS}{\sum (y_i - \bar{y})} \tag{4.5}$$

R^2 measures the variability of Y that can be described using x_i. This error function is explained more in detail in section 4.3.4.

4.3.3 Decision Trees

A decision tree is a classifier, where a recursive division is performed. It is a conditional-based prediction methodology, wherein each node of the tree a probabilistic decision is made. They are commonly used for handling non-linear datasets and can be used both for categorical variable prediction and continuous variable prediction. When it is used for continuous variable prediction, it is called Regression Tree.

This particular type of Decision Tree is built through a process called Binary Recursive Partitioning, where the data is split recursively into different partitions. The algorithm will search for the split that minimizes the sum of squared deviations from the mean. This splitting is performed until each different node reaches a user-specified minimum node size.

Decision trees are made by three types of nodes:

- **Root node**: The "original" node, that has no incoming edges. Is where the tree starts. Represented in blue in the Figure 4.8.
- **Internal node**: Nodes that have in-going and outgoing edges. Represented in red in the Figure 4.8.
- **Decision/Terminal node**: Ending nodes, where the decision is made. Represented in green in the Figure 4.8.

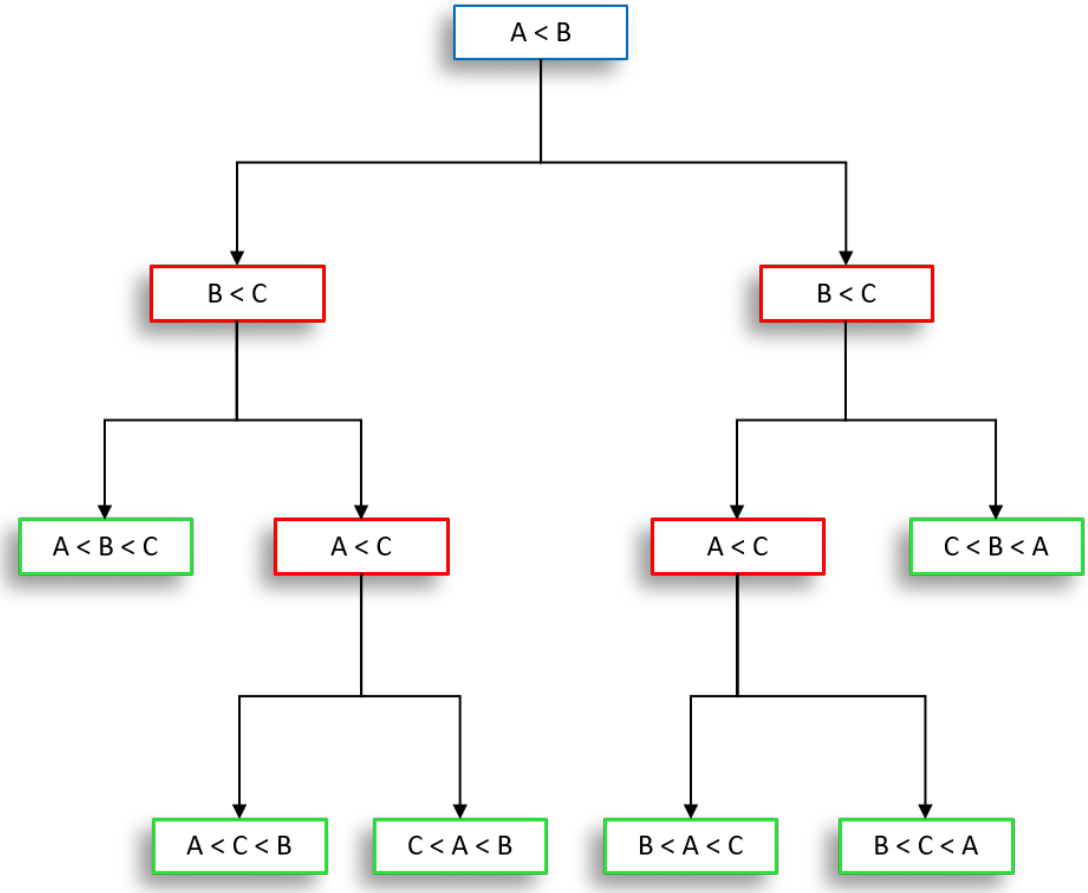

Figure 4.8: Decision tree example

Each internal node splits the instance into multiple edges, following a function that is defined for that node and considers the input attributes.

For its implementation in ML, Decision Tree Inducers are used. These algorithms build automatically a decision tree from a given dataset, finding the optimal decision tree by minimizing the generalization error, the number of nodes, or the tree depth. [11]

4.3.4 Error Functions

Root Mean Squared Error

Root Mean Squared Error (RMSE) is the standard deviation of the residuals in a regression prediction. These residuals represent how far is the regression line to the data points.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{4.6}$$

The equation 4.6 represents the definition of the RMSE, being $y_1, y_2, ..., y_n$ the observed values and $\hat{y_1}, \hat{y_2}, ..., \hat{y_n}$ the predicted values. Dividing by n (number of observations) enables the RMSE to be somewhat independent of the size of the set, just becoming more accurate as the number of observations increases.

RMSE can be seen as the normalized distance between the vector of predicted values and the vector of real values and helps estimate how far off will be the next prediction.

In Data Science and Machine Learning, RMSE has two principal purposes. The first one is to serve as a heuristic for training models. This means that, when training a model, RMSE can be used as a measure of how well the model is adapting and therefore improving it. The second one is to evaluate the accuracy of trained models.

This statistical parameter is a unit-dependent statistical parameter. Depending on the units of the set we are studying, the magnitude of the RMSE will vary. For example, when studying a set of predicted weights, the RMSE will be larger if these weights are measured in grams than if they are measured in kilograms.

In the case RMSE is used as a heuristic in training models, its magnitude will not be as important, as it is used to evaluate the improvement of each iteration. Therefore, it will be studied in relative terms, looking to reduce its value as the model is further developed.

If the RMSE is used to evaluate the accuracy of a trained model, its magnitude will be more relevant. In this case, the user must evaluate if the RMSE is sufficiently small regarding the magnitude of the dependent variable.

R squared

R^2, also known as the coefficient of determination, is a statistical measure that represents the proportion of the variance in the dependent variable that can be explained by the effect of the independent variables. This is represented in the equation 4.7. R^2 measures the relationship between the model and the dependent variable. Looking at equation 4.7, a $R^2 = 0.80$ can be read as 80% of the variance in the model can be explained by the inputs of the model.

$$R^2 = \frac{Variance\ explained\ by\ the\ model}{Total\ variance} \tag{4.7}$$

This statistical parameter is commonly used in Regression models, as studying the influence of independent variables in the prediction is crucial in these models. But low values of R^2 do not always represent a problem in the model, and high values do not represent a good fitting model.

Depending on the nature and complexity of the model, a larger amount of the variation can be inexplicable. On the other hand, high values of R^2 can be miss leading, as R^2 cannot reflect how biased is the model. Therefore, further study of the model must be developed to assess the good fit of the model, using residual plots and other statistical parameters.

4.3.5 AI models

Selecting the adequate algorithm for each application is not a trivial task, and it will largely define the success of the prediction model. When deciding which model to use, there is no definitive guide to decide based on the particular application, although some guiding can be made, as shown in Figure 4.9. The user must decide based on the nature of the data, its structure, the desired prediction, etc. The most important aspects to consider when selecting an algorithm are:

- **Size of the training data**. Although it is always desirable to have a generous amount of data, collecting this data is usually one of the constraints when modeling an AI application. Algorithms with high bias and low variance, like Linear Regression are a good fit when the data set is not sufficiently big.

- **Nature of the dependent variable**. This is one of the first discriminations that must be made when selecting an algorithm. If the dependant variable is a certain class (for example, predicting who will win the next presidential elections), models based on Decision Trees are usually the best fit.
  If, on the other hand, the dependent variable is a continuous value (for example, what will be the stock price of a certain asset next month), Regression-based algorithms will usually be the best choice.

- **Computational time**. Algorithms that provide higher accuracy often require more computational time when training, especially with large sets of data. This would be the case of algorithms such as Neural Network, Random Forest, or Support Vector Machine. More basic algorithms, such as Naïve Bayes or Linear Regression require less training time, being also less precise.

- **Number of features**. If the number of independent variables describing the behavior of the dependent variable is large, this can be miss leading to some algorithms. Models such as Principal Component Analysis (PCA) can be applied to perform a dimension reduction before the actual prediction model.

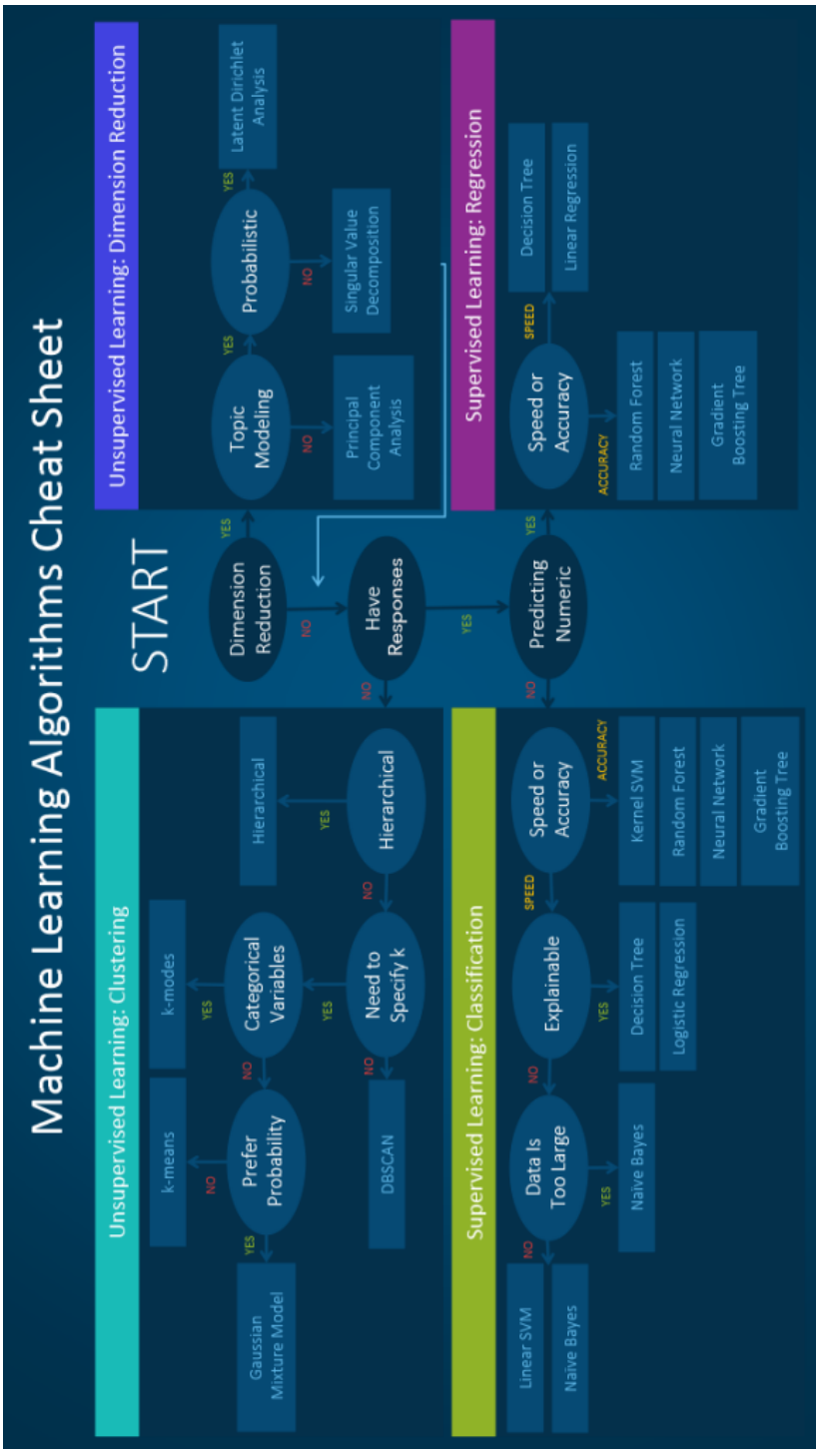

Figure 4.9: Algorithm selection guide [12]

Having this into account, a common practice is to select some candidates and evaluate them in parallel, to choose the best fit. For this particular project, fur algorithms have been selected These are Neural Networks, Support Vector Machines, Gradient Boost and Random Forests. Both Neural Network and Support Vector Machine are Kernel-based algorithms [13], as Gradient Boost and Random Forest are Decision Trees-based algorithms. This will affect the way each algorithm behaves. In the next sections, each algorithm will be described.

Neural Networks

Neural Networks is one of the most popular AI algorithms nowadays, as they provide a wide variety of usages, and can be adapted to almost any application. They are created replicating the structure of the human brain, with individual nodes, called neurons, that interconnect with other nodes to create a complex mesh of correlations. This algorithm's architecture is geared to finding patterns in large and complex sets of data.

These neurons are grouped in layers, where each layer represents one step in the process. This is represented in Figure 4.10. There are three main types of layers:

- **Input layer**. This layer is where the independent variables are introduced to the model. The number of neurons in the input layer is determined by the number of features that are describing the dependent variable.
- **Hidden layers**. These intermediate layers represent most of the computational procedure of the algorithm, where the correlations are defined. The number of hidden layers and the number of neurons in each layer can be defined by the user.
- **Output layer**. In this layer, the final prediction is generated. The number of neurons is defined by the number of outputs predicted.

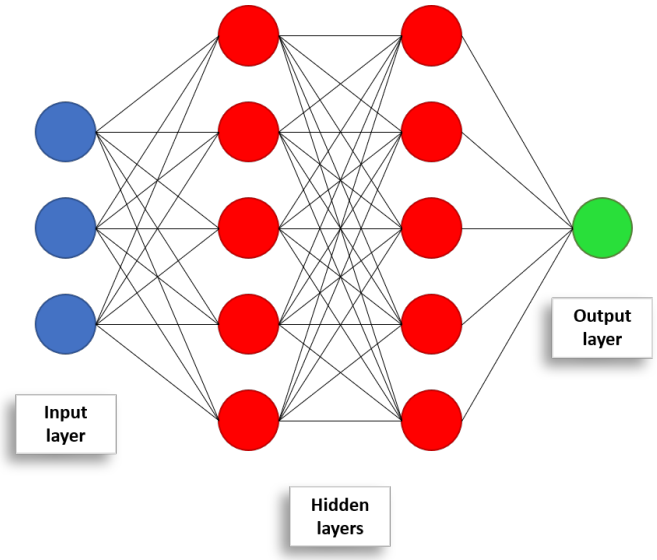

Figure 4.10: Basic Neural Network Architecture

Each node has a connection with all the nodes of the next layer, with a weight assigned to each connection. This weight reflects the impact the output of each node has on the following layer.

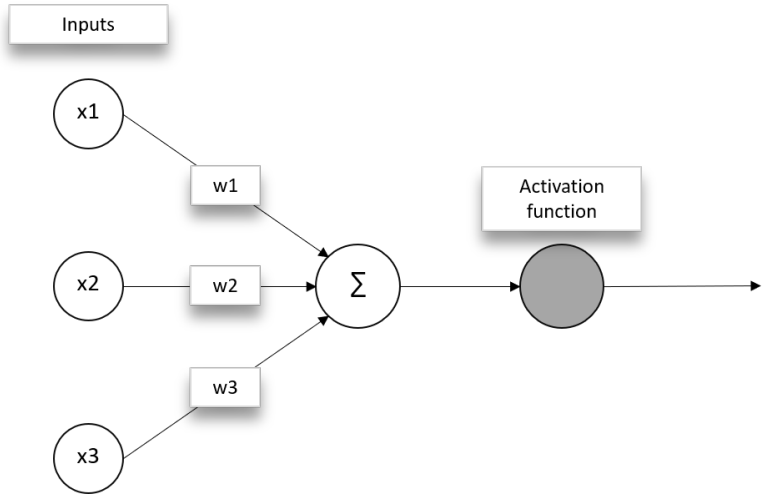

Figure 4.11: Neuron architecture

As represented in Figure 4.11, each node has an activation function, which defines if the node must be activated or not, based on the weighted sum that it receives from other neurons' outputs. The most popular activation functions are:

- **Step function**. In this function, a threshold value is defined. If the weighted sum of the inputs is greater than this value, the node will be activated with value 1. If not, it will not be activated, giving as output value 0.

- **Linear Function**. The output value of the node is generated based on a linear function. Therefore, the output of the node is not confined between any range. A variation of this function is the rectified linear activation function (ReLU), which works the same way as the Linear activation function, but returns a value 0 if the generated output is negative. This modified Linear activation function is one of the most used, due to its versatility and good performance.

- **Sigmoid function**. It is defined by the equation 4.8, and allows the node output to take values between 0 and 1. It is widely used for probabilistic-based predictions.

$$S(x) = \frac{e^x}{e^x + 1} \tag{4.8}$$

The architecture of the NN represented in Figure 4.10 corresponds to a Feed-Forward Neural Network. This architecture only allows the algorithm to learn as it advances on the layers. For more complex implementations, as applications for predicting time series data, audio data, or text data, Recurrent Neural Networks (RNN) is much more convenient. RNN are designed to understand sequential data, where what happened before matters for the actual prediction.

The most important architecture for RNN is the Long Short-Term Memory (LSTM). One of the main problems RNNs have is they tend to give more importance to what just happened before, rather than considering the whole process. This is usually called "short-term memory". Therefore, if the time series/sequence that is the prediction target is long enough, the first part of it will be lost in the process. LSTM is designed to avoid this problem. The neurons for LSTM are built so they learn which data is important, regardless of the length of the data. [14]

Another important aspect to have into account when building a Neural Network is the optimizer that will be used. These optimizers are algorithms that reduce the losses of the Neural Network by changing its parameters, such as weights. There are several types [15], but for this project, the optimizer that will be used is Adam (Adaptive Moment Estimation).

Support Vector Machine

Support Vector Machines (SVM) are algorithms which objective is to find a vector/hyperplane (in a multidimensional plane with N dimensions, being N the number of different features describing the problem) that classifies the data points. This vector/hyperplane is selected between all the possible ones by finding the one that maximizes the distance between both classes (d1 and d2).

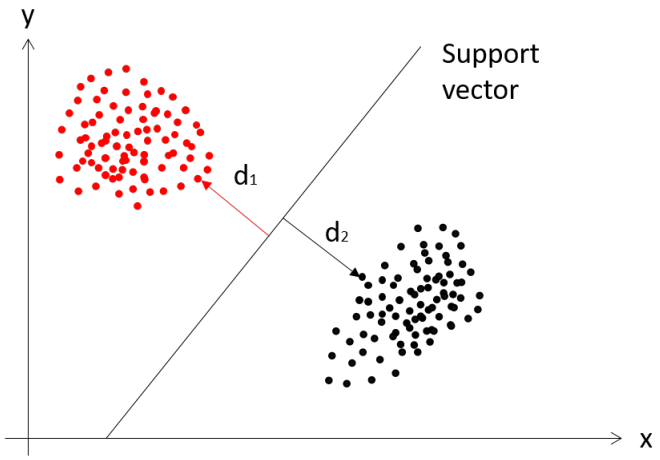

Figure 4.12: SVM classification example

Although SVM is commonly used for classification applications, its use for Regression problems is also extended. In this case, it is called Support Vector Regression (SVR), and the hyperplane is used to fit the data points, considering an error margin defined by the user.

The three main parameters that define an SVR algorithm are:

- **Kernel**. It defines the type of curve/hyperplane that will fit the data.
- **Epsilon** ϵ. Sets the amount of acceptable error when predicting.
- **Regularization parameter C**. Sets a tolerance value for points in the dataset outside the ϵ boundaries.

Gradient Boost

Gradient Boost is not a specific algorithm, but a Machine Learning technique that enables the creation of powerful prediction models using basic Decision Trees, creating a stack of them to have a more precise and complex prediction. The prediction is done in a staggered way, with each tree trying to improve the last tree's prediction. It is commonly used for regression and classification applications. An example of a Gradient Boost base on decision trees is represented in Figure 4.13, where it can be seen how the ensemble method works.

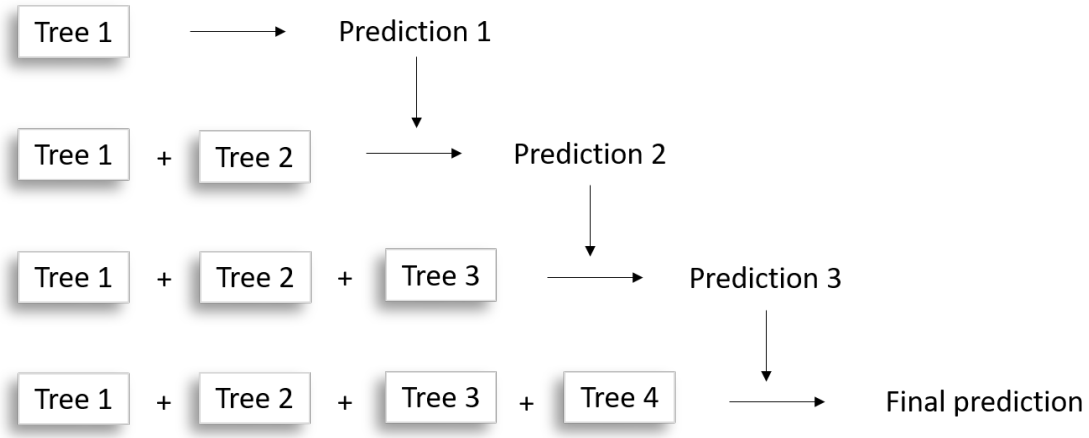

Figure 4.13: Gradient Boost example

Random Forest

Random Forest, as Gradient Boost, is based and formed by multiple decision trees that are stacked. The difference with Gradient Boost is that, in Random Forest, each tree performs the same prediction, and then the prediction with more "votes" is considered as the correct one.

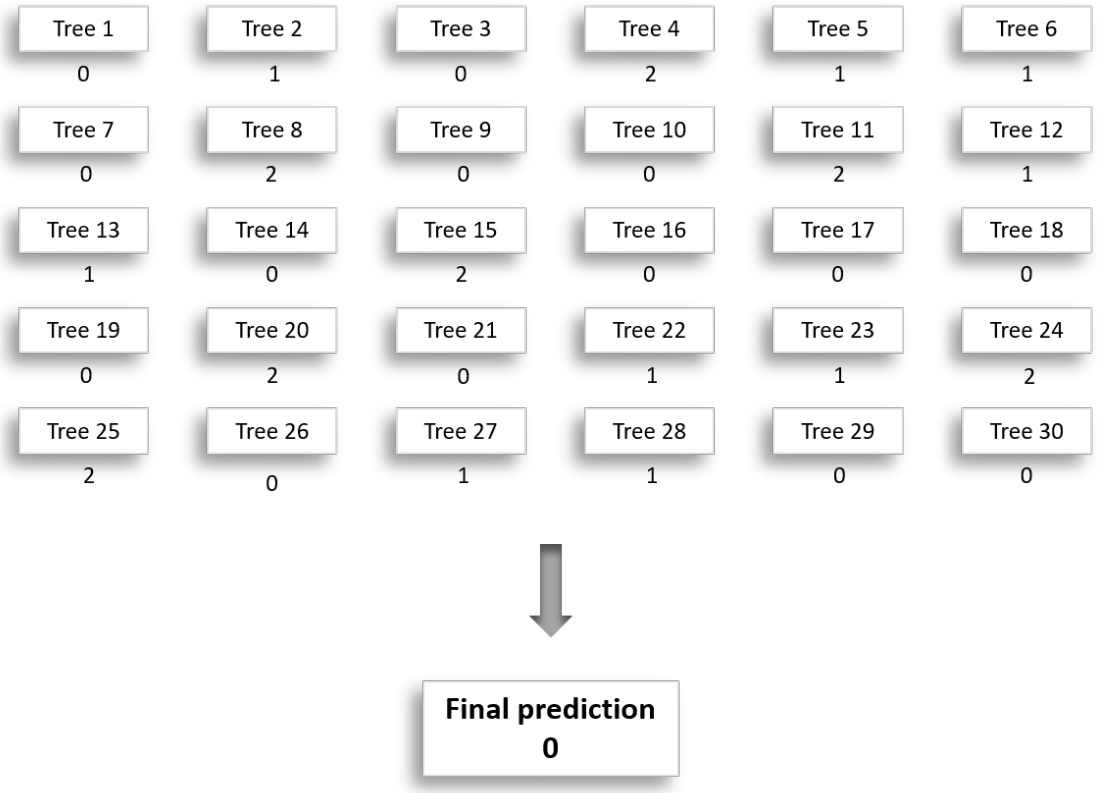

Figure 4.14: Random Forest

The key to the good performance a Random Forest algorithm gives is the lack of correlation between trees, enabling the model to give more accurate predictions than individually or with a more correlated model, as each tree is protected from the possible errors that other trees have.

The randomness of each tree is ensured using a technique called Bagging. As the decision trees are very sensitive to the data that is used to train them, each tree forming the Random Forest is trained with a randomly selected sample from the dataset. This is done so each tree only sees some of the features describing

the model, being these features randomly selected. This ensures a low correlation between trees.

Random Forest is a powerful algorithm, but it is also a very easy-to-implement model. The most important hyperparameter defining it is the number of estimators, being these estimators the Decision Trees. Increasing the number of trees will lower the computational speed, but will increase the precision and also prevent over-fitting.

Chapter 5

Methodology

In this chapter, the process followed to develop the project will be detailed. This process can be divided into two main steps: data analysis & extraction, and AI modeling. All this process has been developed using Python 3.8.3 with the Spyder coding environment [16], using specific packages for improved functionality.

5.1 Data Analysis & extraction

This project relies on the previous work done in the past. Over the last years, numerous projects have been developed at the Volvo Cars Safety Center where crash simulations and tests have been done. All this stored information can now be used to obtain the data to train an AI model, as the resulting output of those crashes is needed. The data extraction process will be designed so new simulations can be introduced to the model.

From this database, the relevant and useful simulations shall be selected, to then process them to extract the data needed for the AI model. This process is represented in Figure 5.1.

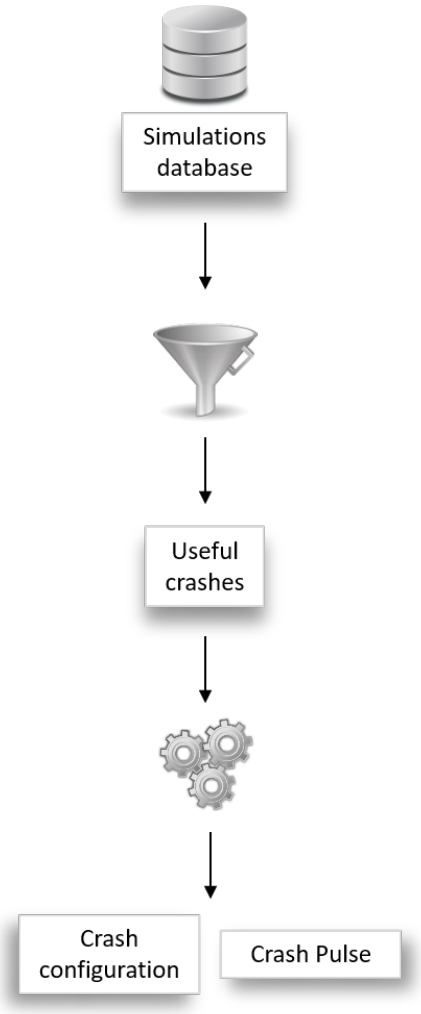

Figure 5.1: Data gathering process.

5.1.1 Data selection

Among all the simulations stored in this database, c2c and c2b have been identified and selected, rejecting those where the minimal requirements to extract the data are not fulfilled. Some of these requirements are having a labeled CoG and an accelerometer for at least one of the objects colliding, having at least 80 ms of recorded results for the crash pulse, or having two voluminous colliding objects. This last restriction is due to a limitation when clustering each object, where the cluster fails when one of the bodies has no volume. This last restriction excludes, for example, a car colliding to a wall, where the wall is defined with a single-node layer.

5.1.2 Data pre-processing

With these valid crashes identified, the needed data shall be extracted from them, adapted to the format that the AI model requires. These simulations are formed by numerous elements and definitions, and they generate a large number of output data. Selecting where to take the information from and validating it is key for a reliable data set. In the next sections, the process to obtain the crash configuration and the crash pulse from each simulation will be detailed.

Crash configuration extraction

The first step for obtaining the crash configuration is to read the simulation files from where the information will be extracted. The information is extracted from output files of the simulation (*d3plot* and *binout*).

- The **d3plot** file contains information about nodes, elements, parts, shells, etc., and their behavior during the simulation (displacements, velocities, etc.). For this project, information about node displacements, as well as part masses and velocities will be used.
- The **binout** file includes a variety of outputs from the simulation, as kinetic energy or node velocities. For this project, information from the **nodout** will be used, where information from specific nodes is recorded. These nodes include CoGs, accelerometers, etc.

The information from these files is read using the lasso.dyna library [17], which is a specific library for reading LsDyna simulation files in Python. This library allows reading the information in a structured way, using cards to define each information type.

When these two files are read, the next step is to determine which objects are colliding. This may seem trivial, but as each object is defined by thousands of nodes, elements, parts, etc; a clustering process must be developed to differentiate each object. The clustering method that has been used is K-Means [10], which is detailed in the K-Means Clustering section. For this specific clustering process, 3 dimensions have been selected:

- Node X coordinate
- Node Y coordinate
- Node ID

A weight has been assigned for the node ID dimension, to reduce its influence in the process. The maximum value of the node ID list has been equalized with the maximum coordinate value. By doing so, the three dimensions will have the same weight on the cluster. This is represented in tables 5.1 and 5.2 (note that the values represented in these tables do not correspond to real values). The number of clusters is defined as 2 (number of colliding objects).

Node number	X-coord	Y-coord	Node ID
Node 1	135.45	-2004.53	1
Node 45234	526.23	153.65	45234
Node 2343244	2034.34	-1121.43	2343244
...	...	...	...
Maximum values	3025.65	3185.54	58523443

Table 5.1: Dimensions of the cluster without weighting

Node number	X-coord	Y-coord	Node ID
Node 1	135.45	-2004.53	1
Node 45234	526.23	153.65	853.51
Node 2343244	2034.34	-1121.43	2543.43
...	...	...	...
Maximum values	3025.65	3185.54	3185.54

Table 5.2: Dimensions of the cluster after weighting

As part of this clustering process, and to correctly detect the FPOC, an iterative process has been developed to detect the last timestep of the simulation before the collision. This process has been designed to start from timestep 0, cluster both objects, detect the boundaries and then check if they are intersecting. If not, the loop continues and repeats the same process for the next timestep. When it detects intrusion between boundaries, it returns the immediate previous timestep, and this one will be used to do the final cluster and detect the FPOC.

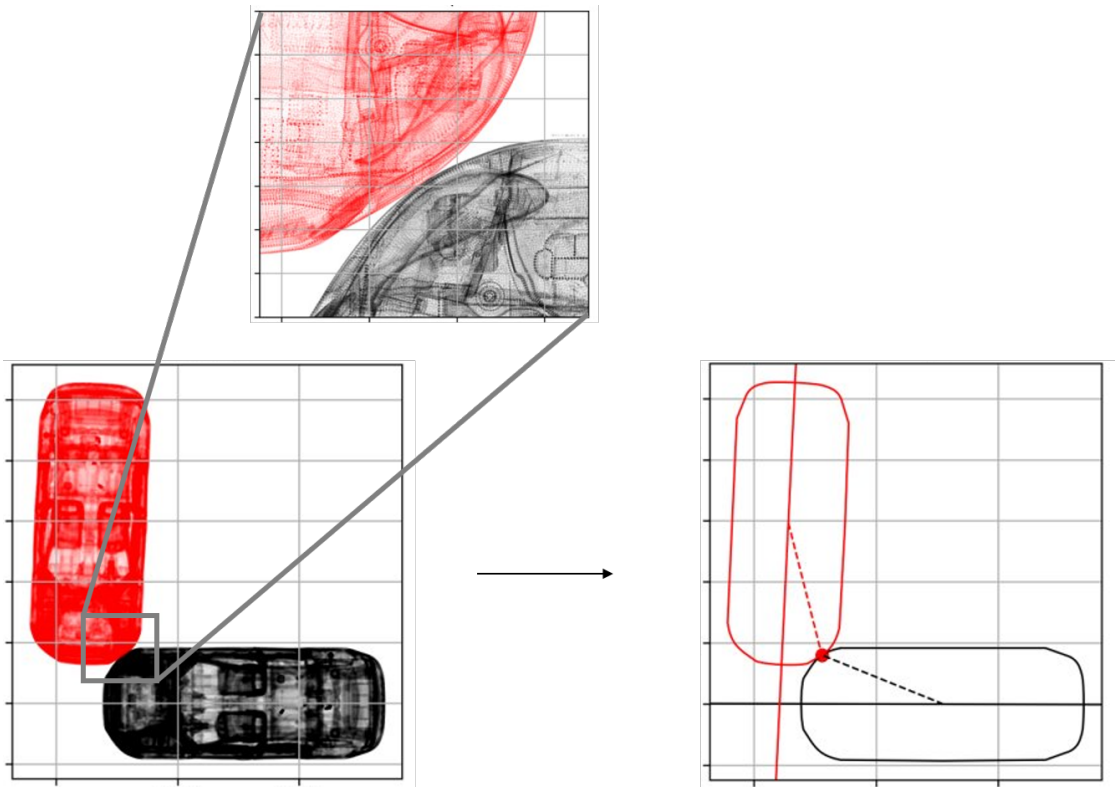

Figure 5.2: Object detection process

When the final cluster is obtained, to find the FPOC, the boundary of each cluster must be found. The first step is to create a Convex Hull [18], to later on creating a polygon [19] around this hull. With both polygons, the minimum distance can be found using the built-in function in Shapely objects [19]. This Shapely object will also be useful to perform the vehicle normalization. This is done by reducing the vehicle/object dimensions to a 1x1 square. This normalization allows the comparison of crash configurations between vehicles with different dimensions [6].

The vehicle normalization is the last step before calculating the positional (HCPA, OCPA, OYA) angles described in the Crash Configuration section. With these angles defined, the last two steps are to extract the velocities and the masses of both vehicles.

As for the velocities, the information is extracted from the **nodout** file. If a CoG node is identified within each vehicle boundary, the velocity is directly extracted from these nodes. If this is not the case and one of the CoG is missing, the velocity for that vehicle is extracted from nodes in the **nodout** that are within the boundary. A sample of these nodes is selected, to ensure the information is extracted correctly.

The mass information is stored in the **d3plot** file, and it is assigned to the parts defined in the model. As the vehicle identification has been done regarding the nodes, it is not known which part belongs to which vehicle. For this reason, a new clustering process must be developed.

In this case, there will be four dimensions:

- Part ID
- Part mass
- Part Velocity in X coordinate
- Part velocity in Y coordinate

Each cluster is then compared with the already known vehicle velocities, to define which cluster is which vehicle. Once this is known, the mass of each part belonging to each vehicle is summed up to obtain the vehicle's total mass. This procedure is sensitive to added masses, as dummies.

With the velocities and masses, all the information regarding the Crash Configuration is extracted. As the last step, an angle decomposition must be performed. This decomposition in X-Y coordinates will be applied for the three angles (HCPA, OCPA, and OYA).

This is graphically described in Figure 4.4. Two angles that represent practically the same point (P1 & P2) have a widely different numerical representation (178 & -178). This can be miss leading for the AI model.

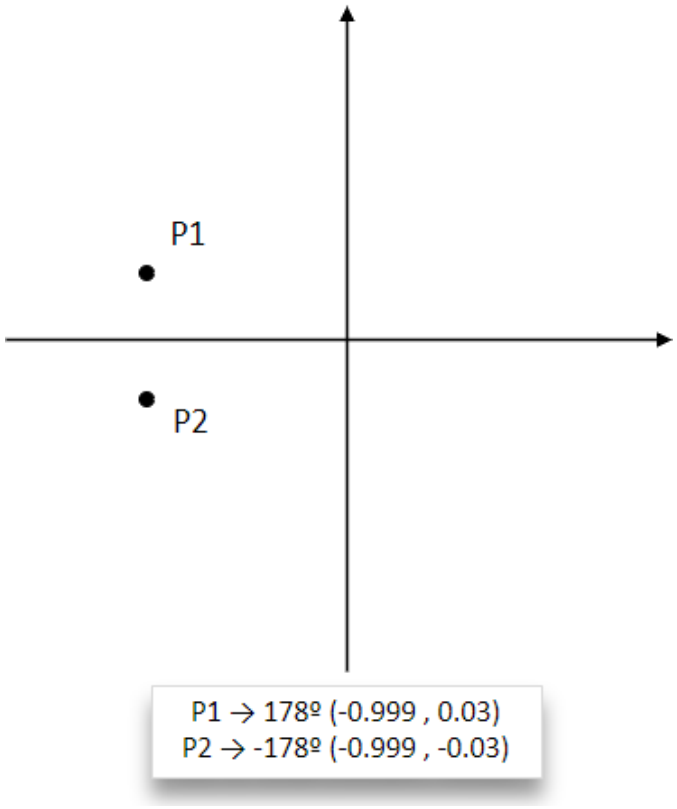

Figure 5.3: Angle decomposition

After the decomposition of HCPA, OCPA and OYA, the final Crash Configuration definition consists of 10 parameters, all of them numerical and continuous. This is represented in Figure 5.3.

HCPA-X	-0.98
HCPA-Y	0.2
OCPA-X	0.98
OCPA-Y	-0.2
OYA-X	1
OYA-Y	0
Host speed	13.5
Opponent speed	21
Host mass	2354.5
Opponent mass	2154.2

Table 5.3: Crash Configuration parameters

All this process of Crash Configuration extraction has been executed automatically for each simulation that fulfilled the requirements described in section 5.1.1, and the Crash Configuration of each simulation has been stored in a database, where later on the information of the Crash Pulses will be annexed.

With this database fully defined, the division to select the frontal crashes has been carried out. In order, to do so, a limit has been set in the HCPA parameter, where all the simulations where this parameter is within the ranges of [-45,45] and [135,225] have been identified as frontal/rear-end crashes, as explained in section 4.1.1.

Crash Pulse extraction

The remaining part of the data analysis process is to extract the Crash Pulse of each simulation. This Crash Pulse is formed by 3 axis translation and rotation time series, as described in section 4.2. The information for the crash pulse is extracted using a pre-existing script. This script computes the Crash Pulse in the shape of velocities, using the information stored in the **nodout** file for the accelerometers positioned in the vehicle. It is important to use the information in the global coordinates system, as these accelerometers compute the information both in global and local coordinates. This velocity crash pulse can be integrated or derived to obtain displacements or accelerations, respectively.

However, this script has been modified to fit the purpose of this project. The input parameters for the script are the ID of the accelerometers with global coordinates information (3 in total), the read file path, the writing file path, and a pid parameter. The 3 used accelerometers are positioned in a region of the vehicle body that is not deformed during the crash.

The script has also been modified to generate a reduced number of points for the Crash Pulse time series, with an x100 reduction. The output of the script is a .k file with the information for the 6 time series, in this case, translational accelerations and rotational positioning. The models have been tested with accelerations, velocities, and displacements, both for translations and rotations, concluding that translational accelerations (m/s^2) and rotational displacements (rad) are the best ones for predicting.

However, these extracted Crash Pulses are not suitable yet for the model. This information must be adapted to a format the AI model can interpret. Principal Component Analysis (PCA), Symbolic Aggregate Approximation (SAX) and frequency analysis have been considered as methods to perform a dimensional reduction of the time series. However, due to limitations of replicability, and looking for a data structure that can be easily introduced in an AI algorithm, it has been decided to perform the reduction of the Crash Pulse, dividing the time series in time steps of 5 ms. The mean value of each timestep's data will be calculated. This allows not only to reduce the number of data points, which will reduce the complexity of the needed model, but to make the data independent from time, as each value is assigned to a particular step, equal for each simulation. Therefore, each data point will be uni-dimensional.

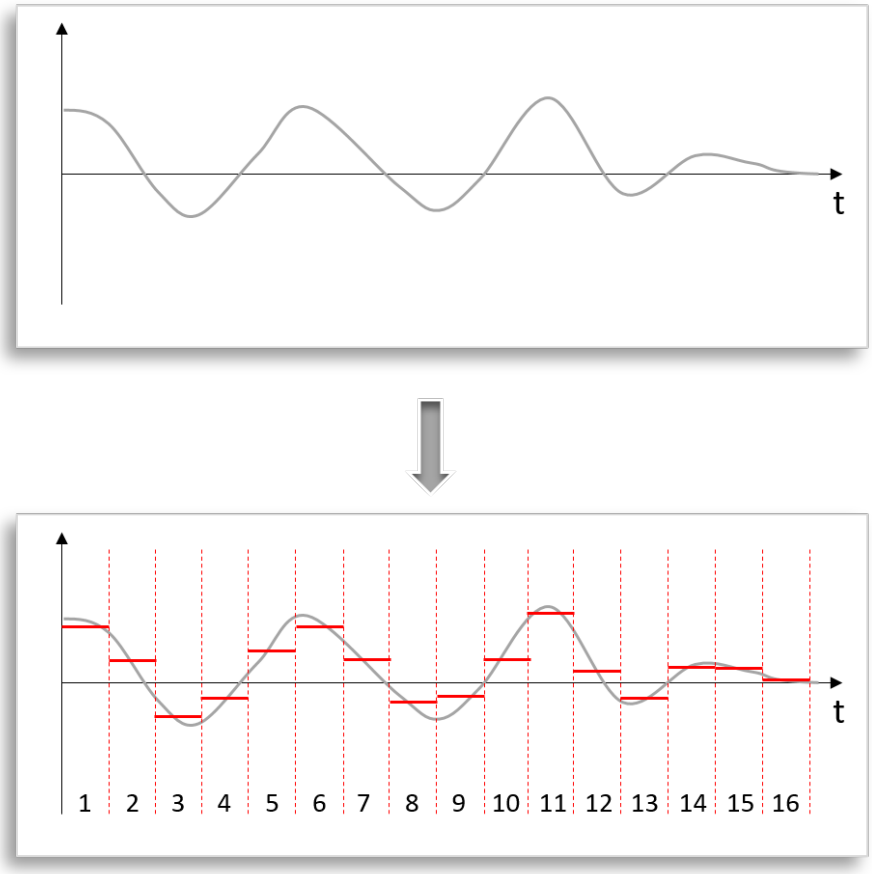

Figure 5.4: Dimensional reduction of the data

After performing this reduction, each simulation crash pulse is described by 96 values, being each curve described by 16 values. This is represented in Table 5.4.

Steps	X-axis tr	Y-axis tr	Z-axis tr	X-axis rot	Y-axis rot	Z-axis rot
Step 1	#1	#17	#33	#49	#65	#81
Step 2	#2	#18	#34	#50	#66	#82
Step 3	#3	#19	#35	#51	#67	#83
Step 4	#4	#20	#36	#52	#68	#84
Step 5	#5	#21	#37	#53	#69	#85
Step 6	#6	#22	#38	#54	#70	#86
Step 7	#7	#23	#39	#55	#71	#87
Step 8	#8	#24	#40	#56	#72	#88
Step 9	#9	#25	#41	#57	#73	#89
Step 10	#10	#26	#42	#58	#74	#90
Step 11	#11	#27	#43	#59	#75	#91
Step 12	#12	#28	#44	#60	#76	#92
Step 13	#13	#29	#45	#61	#77	#93
Step 14	#14	#30	#46	#62	#78	#94
Step 15	#15	#31	#47	#63	#79	#95
Step 16	#16	#32	#48	#64	#80	#96

Table 5.4: Crash pulse description

5.2 AI modelling

5.2.1 Dataset division

Now that the data has been collected and pre-processed, the dataset is ready to be introduced to an AI algorithm. To do so, the dataset must be split into training and test sets.

- **Training set**. This set will be introduced to the supervised algorithm with both Crash Configuration (input) and Crash Pulse (output). With this set, the model will learn what output is expected from the given input, adjusting the weights and internal parameters of the algorithm.
- **Test set**. From the base dataset, a selection of simulations must be extracted, to be introduced as a validation set. This set will be introduced to the model without the Crash Pulse (output) information. The model will predict the values of the output, so they can be compared with the actual output. This will help evaluate the precision of the model, comparing prediction with actual values.

There are different strategies when dividing the dataset, but the most common, and the one used in this project, is to randomly divide the dataset, using a certain percentage of the dataset for training and the rest for test. In this project, the set has been divided 80% training and 20% test. This random division helps to ensure that neither set is biased regarding the base dataset.

This splitting has been performed using Sklearn functionality, where the dataset is introduced, as well as the percentage of the dataset that is designated to testing, and a randomizer parameter, that shuffles the dataset before splitting. [20]

5.2.2 AI algorithms

Four different AI algorithms have been tested, to determine which one is the most suitable for the prediction of the Crash Pulse. The structure of these four algorithms is explained in detail in section 4.3.5.

Now, how each algorithm has been structured and created will be detailed. Note that the process to find the best structure has been carried out via trial-and-error, as there is no a "perfect" solution for each application.

The goodness of fit will be evaluated using R^2 and RMSE metrics, whose operation and meaning is detailed in section 4.3.4

Neural Networks

Neural Networks general information is detailed in section 4.3.5.

Two different models have been created, one for translations and one for rotations. Their architecture is formed by:

- **Input layer**. The number of neurons in this layer is determined by the number of inputs that will be introduced to the model. In this case, 10 neurons.
- **1st hidden layer**. Dense layer formed by 256 neurons, with a ReLU activation function.
- **2nd hidden layer**. Dense layer formed by 128 neurons, with a ReLU activation function.
- **Output layer**. As in the input layer, the number of neurons is determined by the number of desired outputs. The activation function is defined as Linear.
- **Compiler**. Here is where the loss function and the optimizer are defined. The optimizer has been set as Adam [21], and the loss metric for the loss function Mean Squared Error (MSE).

The model has been trained with 1500 epochs (hyperparameter that defines the number of times the algorithm will go through the entire dataset). [22]

For better performance, normalization has been performed, subtracting the mean of each parameter and dividing by the standard deviation. This is represented in equation 5.1, where index i represents each parameter and index j represents each simulation or data point.

$$X_{i,j} = \frac{X_{i,j} - \overline{X_i}}{X_{dev\ i}} \tag{5.1}$$

The algorithm has been created using Keras library for Python.[23]

Support Vector Regression

Support Vector Machines general information is detailed in section 4.3.5.

The input has been normalized, as explained in the Neural Networks section.

The structure of the SVR model has been defined as a recursive process. As SVR only allows to do one prediction at a time, 96 models shall be created to predict the 96 points that describe the Crash Pulse.

A common model architecture has been defined, using a 3$^{\text{rd}}$ degree polynomial and a $\epsilon = 1 \times 10^{-8}$. This architecture will be replicated for each model. It has been created using Sklearn specific class for Support Vector Regression. [24]

The strategy for building the different models could have been to create individual models, each one predicting one output. This would have resulted in models with no correlation. To avoid this, a recursive model has been created. This recursive approach has been used for diverse applications [25][26][27]. This way, each consecutive model will have the information from previous steps, being able to "learn from the past". The two different strategies are represented in Figures 5.5 and 5.6. Using the recursive strategy, each prediction will be added to the input dataset for the next model.

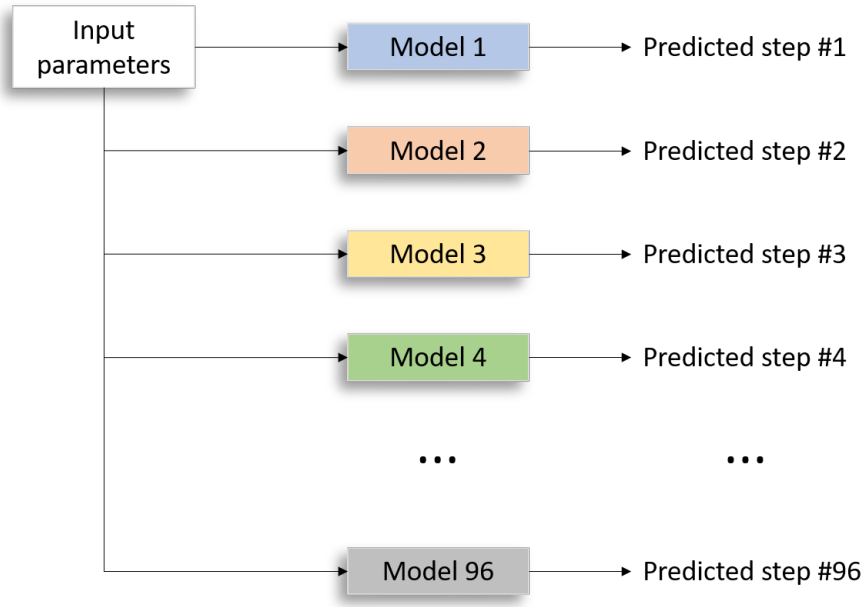

Figure 5.5: Direct strategy for multi-step prediction example

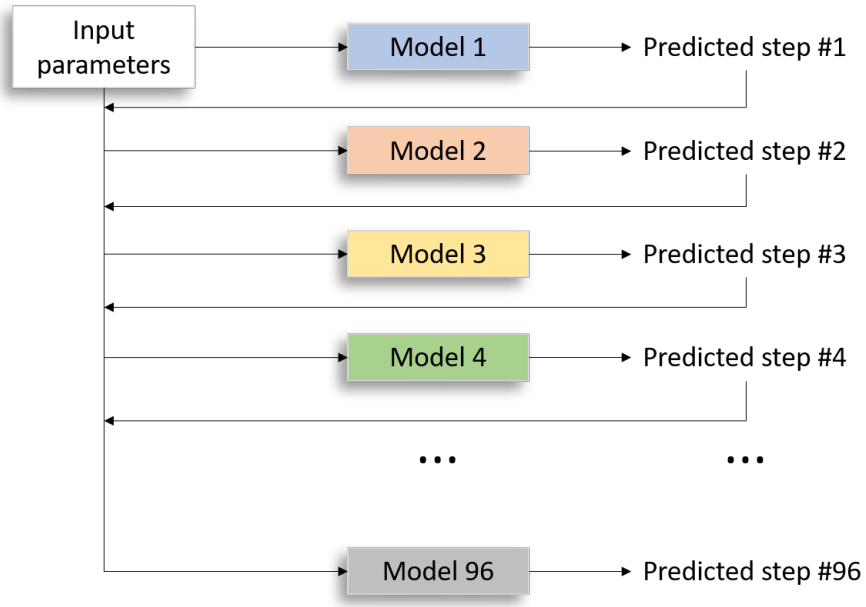

Figure 5.6: Recursive strategy for multi-step prediction example

Gradient Boost

General information about Gradient Boost is detailed in section 4.3.5.

As for the SVR, the Gradient Boost architecture only allows predicting one variable per model. In this case, unlike in SVR, a direct multi-step prediction strategy has been followed. This has been done by creating a stack of models, using MultiOutputRegressor class from Sklearn [28], each one trained individually with the same architecture.
The number of estimators has been set to 100, being their maximum depth 4, the learning rate 0.05 and the loss function to be optimized Least Squares. The model has been created using Sklearn class for Gradient Boosting Regressor [29]. Using MultiOutputRegressor, the fit of the algorithm and its validation is done directly to the stack of models, not having to fit every individual algorithm manually.

Random Forest

General information about Random Forest is detailed in section 4.3.5.

Random Forest Regressor allows predicting multiple outputs. A unique model has been tested but results for rotations were poor. Therefore, two different models have been created, one for translations and one for rotations. Each model has been defined with an adequate architecture, consisting of 50 estimators or decision trees for the translations model and 150 estimators for the rotations model, each one with a maximum depth of 15. Normalization of the input is not needed for this algorithm. The model has been created using Sklearn class for Random Forest Regressor [30].

Chapter 6

Results

6.1 Dataset

As described in Figure 5.1, the first step of the data extraction process was to select the simulations, from the whole available set of simulations, that complied with different requirements. After doing this selection as stated in Section 5.1.1, a total of 284 frontal crash simulations have been selected. These simulations include both car-to-car and car-to-moving barrier simulations, as both are similar dimensional-wise.

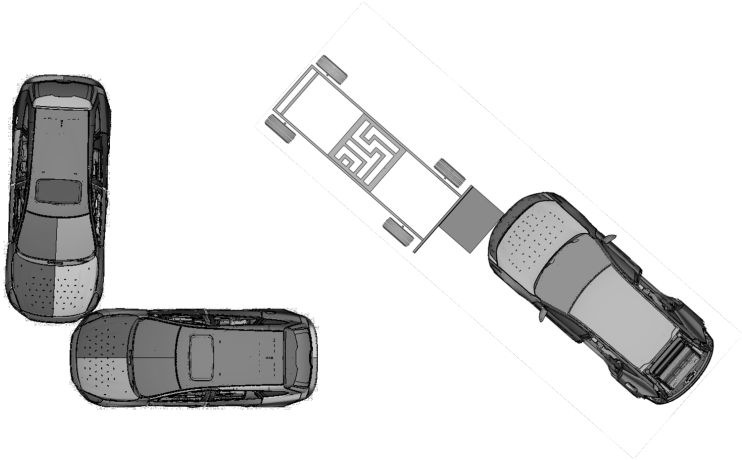

Figure 6.1: Car-to-car and car-to-moving barrier simulations

For these 285 simulations, the process of Crash Configuration extraction has been performed, resulting in the dataset represented in Figure 6.2, being the x-axis the different values of HCPA and the y-axis the values of OCPA.

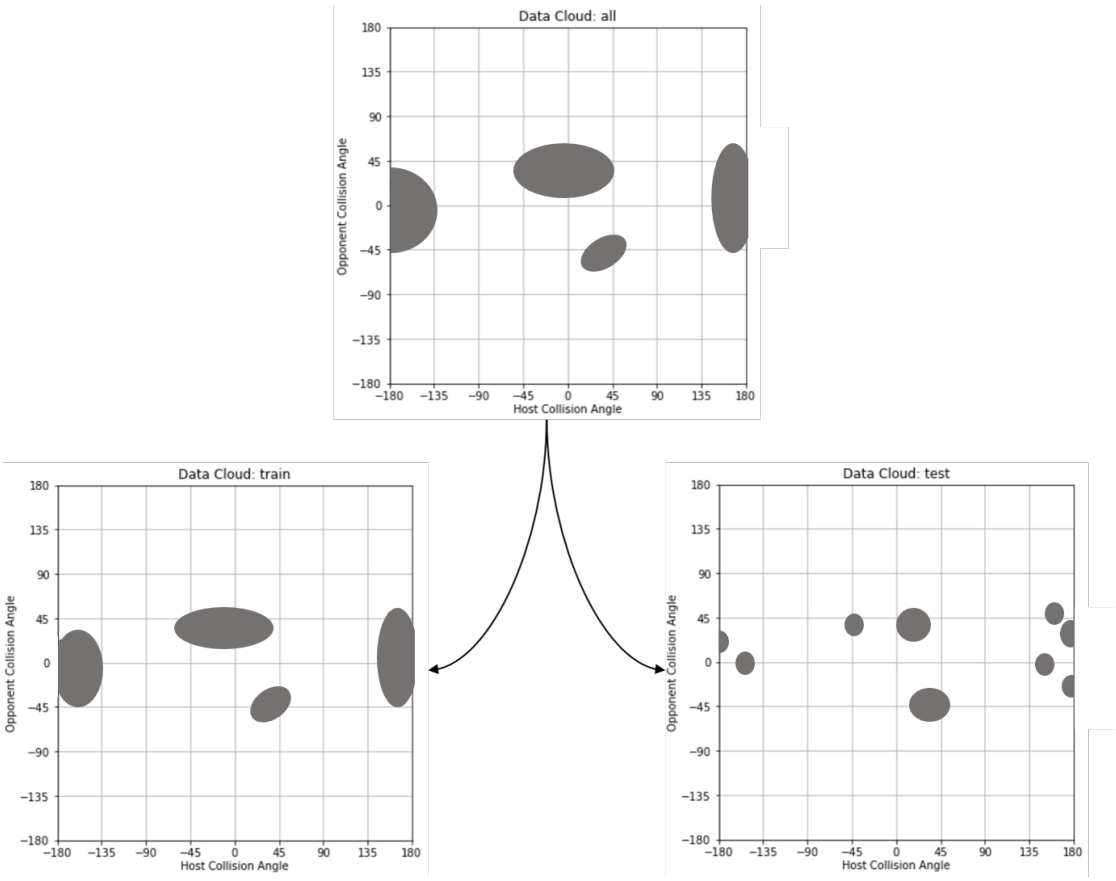

Figure 6.2: Resulting data cloud and splitting

Although it is not visible in Figure 6.2, simulations with the same HCPA and OCPA configuration may vary in terms of OYA, speeds, or masses. This will result in unique simulations, with unique results. In Figures 6.3 and 6.4 the number of data points regarding speeds and masses both for Host and Opponent is represented.

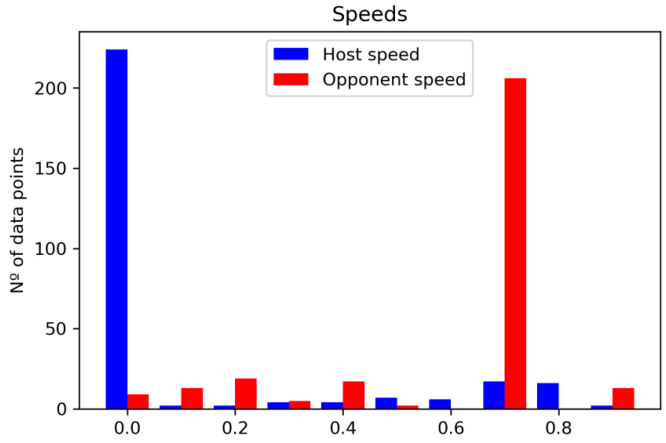

Figure 6.3: Frequency analysis of Host and Opponent speeds in dataset

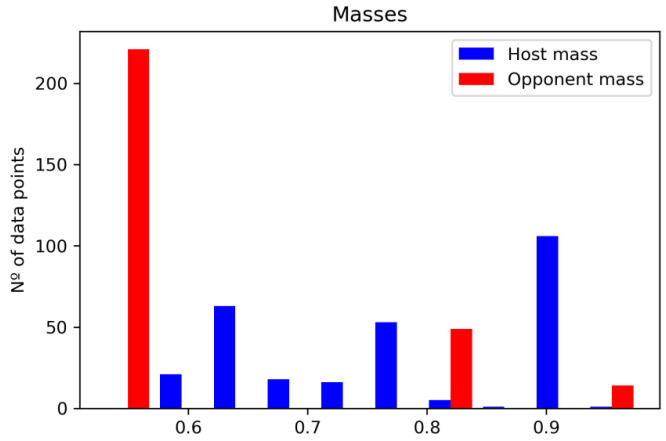

Figure 6.4: Frequency analysis of Host and Opponent masses in dataset

In parallel with the crash configuration extraction, the crash pulse has also been extracted from each simulation and processed to obtain the describing 16 steps. Each simulation crash pulse being represented by 6 curves, the resulting data for each one is 96 uni-dimensional points, that have been stored with the crash configuration in a dataset that combines input/output information of each simulation.

After the dataset was created, the splitting into train and test sets has been performed as described in Section 5.2.1. These split sets are also represented in Figure 6.2. The train set is formed by 232 simulations (81.7% of the dataset), while the test set is formed by 52 simulations (18.3% of the dataset).

6.2 AI implementation

The four models described in section 5.2.2 have been tested and compared using RMSE and R^2 error metrics. These parameters have been computed for each step, and the average error has also been computed for translations and rotations.

The graphs representing the error metrics for each step are attached in Appendix B. The average error metrics for each model are detailed in Table 6.1.

	Translations		Rotations	
Algorithm	RMSE	R^2	RMSE	R^2
Neural Network	0.020	0.728	0.0024	0.103
Support Vector Machine	0.017	0.805	0.0023	0.319
Gradient Boost	0.019	0.747	0.000126	0.834
Random Forest	0.0189	0.751	0.00229	0.937

Table 6.1: Average error metrics for each model

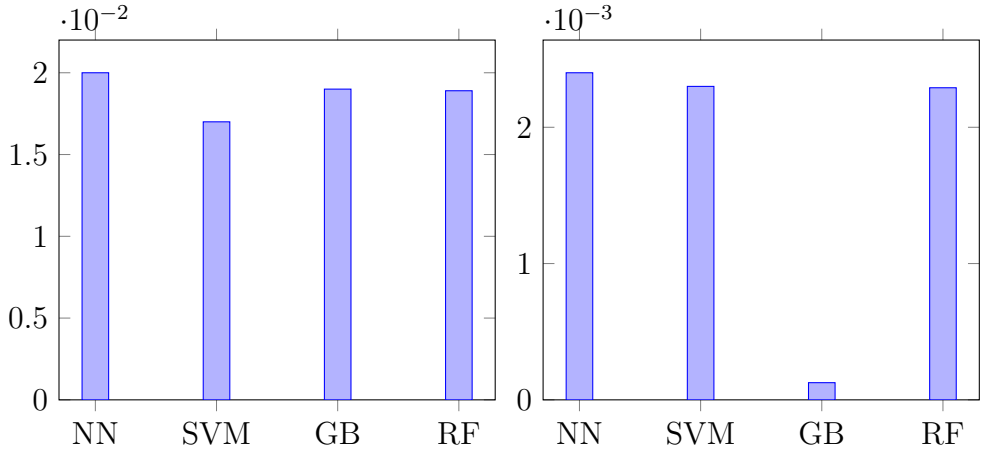

Figure 6.5: Comparison of RMSE for translations (left) and rotations (right)

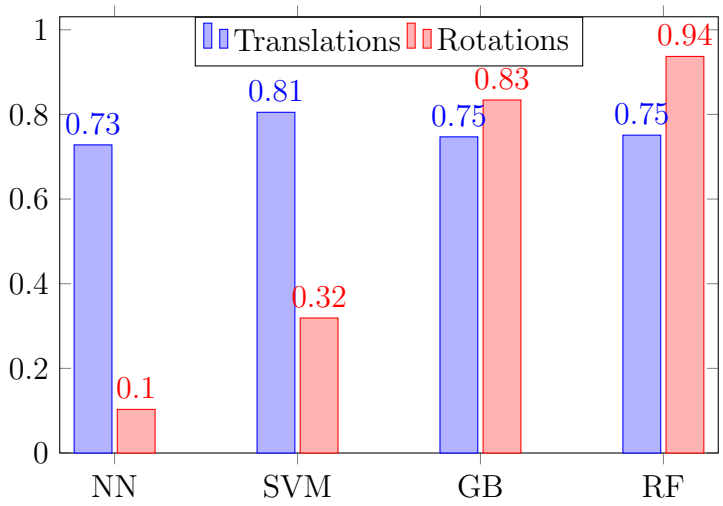

Figure 6.6: Comparison of R^2

Chapter 7

Conclusions

Looking at the Results, some conclusions can be drawn. The first and main one is that, when building an AI algorithm, there is no perfect model. Evaluating the performance of a model is a complex task, and usually a subjective one. Choosing the adequate metrics to evaluate the performance will affect the analysis, and may result in different conclusions.

Another important factor is the dataset. Building a dataset that is representative enough but not biased is key for creating a good AI algorithm. In these terms, the dataset obtained in this project, as represented in Figures 6.2, 6.3 and 6.4, combines multiple and diverse configurations. Nevertheless, some clustering can be seen regarding HCPA, OCPA, speeds, and masses. This will affect the accuracy of the predictions, especially for crash configurations with large differences in these parameters compared to the dataset. This will be especially important if Lateral Crash Configurations are predicted. As no information from similar Crash Configurations is recorded in the dataset, the models would perform poorly, as there is no similar information to use as a base.

In the frequency analysis of the speeds and masses, it can be seen that there are a large number of simulations that have the same values for these parameters. This can be explained by the fact that many of these crash simulations represent standardized crashes, and the vehicle models are constrained to Volvo cars.

Regarding the AI implementation, looking at the error values in the Results chapter and Appendix B, the evaluation of the performance of each algorithm can be carried out. Looking at the predictions for the Translations, the four algorithms have similar performance, being the Support Vector Machine the model with less RMSE (0.017), and the largest R^2 (0.805). Therefore, this model can be defined as the best fit for Translations prediction. For the Rotations models,

the models based on Decision Trees (Gradient Boost and Random Forest) outperform the kernel-based models by a large extent (0.834 & 0.937 vs. 0.103 & 0.319). Therefore, it can be stated that for the Rotations prediction, a model based on Decision Trees is more suitable. Between both Gradient Boost and Random Forest, a higher R^2 in the Random Forest (0.937 vs. 0.834) can lead to the thought that this model is predicting better. This assumption is not always correct, as overfitting of the model can lead to these high values of R^2. In addition, the mean RMSE for Gradient Boost is significantly better compared to the mean RMSE for Random Forest (0.000126 vs 0.00229). Therefore, a more detailed analysis must be done.

Looking at the predictions step-by-step in Appendix B, the R^2 values for Random Forest are much more consistent compared to the values for Gradient Boost, being each step over 0.8. For the Gradient Boost algorithm, inconsistencies in Y and Z axes can be seen, even going below 0 for the last step in Y-axis. Negative values of R^2 mean that the prediction is done just by considering the mean values of the points learned during the training phase.

Due to this inconsistency in R^2 values in late stages for the Gradient Boost, the Random Forest has been considered the best fitting algorithm for Rotations. Therefore, the selected algorithms are **Support Vector Machine Regressor** for Translations and **Random Forest Regressor** for Rotations.

Although the Support Vector Machine Regressor has been created as an iterative process combining Translations and Rotations, where each model is dependent on the previous predictions, as the Translations of the Crash Pulse are predicted before the Rotations, cropping the model to only predict the Translations will not affect the predictions.

Chapter 8

Limitations & Future Work

During the development of this project, different problems and limitations have been encountered. In this chapter, the main ones will be detailed, also commenting on possible future work.

- As it can be seen in Figures 6.2, 6.3 and 6.4, the dataset obtained for this project can be considered as clustered. More simulations may be needed to fill the dataset and have a more consistent data cloud. This dataset is also restricted by the fact that only Frontal Crashes are considered. Although frontal and lateral crashes are very different in terms of crash dynamics, introducing lateral crashes could help to complete the dataset, helping therefore the predictions for crashes that are close to the limit marked between frontal and lateral.

- The length of a crash pulse depends largely on the type of the crash. Depending on the Crash Configuration, the important part of the crash pulse can be at 40 ms or 120 ms. For this project, the Crash Pulses needed to be cropped to 80 ms due to the usage of Supervised Learning algorithms. Unsupervised Learning algorithms have gained importance in the last years and could be a good solution, as no need for output information for training is needed.

- Crash Pulses are complex, as well as their relationship with the severity of the crashes. The time series extracted from the simulations are formed by thousands of points. It would be ideal to predict directly these time series, but this fell out of the scope of the project. For easier implementation, the Crash Pulses have been reduced to 16 intervals. This reduction is enough when covering the objectives of the project, but for wider implementation, a more precise representation of the Crash Pulse may be necessary.

- As the information has been all extracted from a single manufacturer, differences in Crashworthiness and general behavior of the vehicle between vehicles from different manufacturers and not considered in this project. Including crashes from other manufacturers could add more richness to the model. For this, an additional parameter describing the manufacturer, or even the vehicle model, could be needed. From a generalist point of view, completing the definition of the Crash Configuration with additional parameters would enrich the model.
- Due to time limitations, no information from physical tests has been extracted, as their information structure varies largely from the simulations. These tests would help validate the model with real-world data, and could also provide additional information.

Bibliography

[1] WHO. "Road traffic injuries". In: (2016). URL: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries.

[2] Martin Östling. "Predicted crash configurations for Autonomous Driving vehicles in mixed German traffic for the evaluation of occupant restraint system". In: (2019).

[3] Martin Östling. "Passenger car safety beyond ADAS: Defining remaining accident configurations as future priorities". In: (2019).

[4] Gerald Joy Sequeira. "Evaluation and characterization of crash-pulses for head-on collisions with varying overlap scenarios". In: (2020).

[5] Linus Wågström. "Integrated Safety: Establishing Links for a Comprehensive Virtual Tool Chain". In: (2019).

[6] Alexandros Leledakis. "A method for predicting crash configurations using counterfactual simulations and real-world data". In: (2020).

[7] Zhiqing Cheng. "Optimal Crash Pulse for Minimization of Peak Occupant Deceleration in Frontal Impact". In: (2005).

[8] Wesley Grimes. "The Effect of Crash Pulse Shape on Occupant Simulations". In: (2000).

[9] Patil Kantilal. "Crash Pulse Characterization to Minimize Occupant Injuries in Offset Frontal Crash". In: (2017).

[11] Oded Maimon. *The Data Mining and Knowledge Handbook*. 2005, pp. 165–192.

[12] Hui Li. "Which machine learning algorithm should I use?" In: (2020). URL: https://blogs.sas.com/content/subconsciousmusings/2020/12/09/machine-learning-algorithm-use/.

[13] Klaus-Robert Müller. "An Introduction to Kernel-Based Learning Algorithms". In: (2001).

[14] Michael Phi. “Illustrated Guide to LSTM’s and GRU’s: A step by step explanation”. In: (2018). URL: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[15] Sanket Doshi. “Various Optimization Algorithms For Training Neural Network”. In: (2019). URL: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6.

[21] Diederik Kingma. “Adam: A method for Stochastic Optimization”. In: (2015).

[22] Jason Brownlee. “Difference Between a Batch and an Epoch in a Neural Network”. In: (2018). URL: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/.

[25] Souhaib Ben Taieb. “Recursive and direct multi-stepforecasting: the best of bothworlds”. In: (2012).

[26] Arun Venkitaraman. “Recursive prediction of Graph Signals with Incoming Nodes”. In: (2019).

[27] Puning Xue. “Multi-step ahead forecasting of heat load in district heating system susing machine learning algorithms”. In: (2019).

Python documentation

[10] *Scikit KMeans Documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

[16] *Spyder*. URL: https://www.spyder-ide.org/.

[17] *LASSO Python Library*. URL: https://lasso-gmbh.github.io/lasso-python/build/html/dyna/dyna.html.

[18] *Scipy Convex Hull Documentation*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html.

[19] *Shapely Documentation*. URL: https://shapely.readthedocs.io/en/stable/manual.html.

[20] *Sklearn Train and Test Splitting*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[23] *Keras*. URL: https://keras.io/guides/.

[24] *Sklearn SVR Documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html.

[28] *Sklearn MultiOutputRegressor Documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html.

[29] *Sklearn Gradient Boosting Regressor Documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html.

[30] *Sklearn Random Forest Regressor Documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

Appendix A

Alignment with the SDGs

Mobility is one of the key features in a globalized world, and for it to be effective, it must be safe and reliable.

Considering the SDGs (United Nations, 2015) and their purposes and scopes, this project is aligned with several of them. The main ones are the following:

- **Goal 3** - Good health and well-being. This SDG has as primary objectives to ensure healthy lives and to promote well-being for all at all ages. Therefore, one of the main tasks is to reduce the number of global deaths, and specifically the road traffic accidents, as defined in target 3.6.
- **Goal 11** – Make cities and human settlements inclusive, safe, resilient, and sustainable. The target 11.2 is to provide access to safe, affordable, accessible, and sustainable transport systems for all, improving road safety.

Appendix B

B.1 Neural Networks

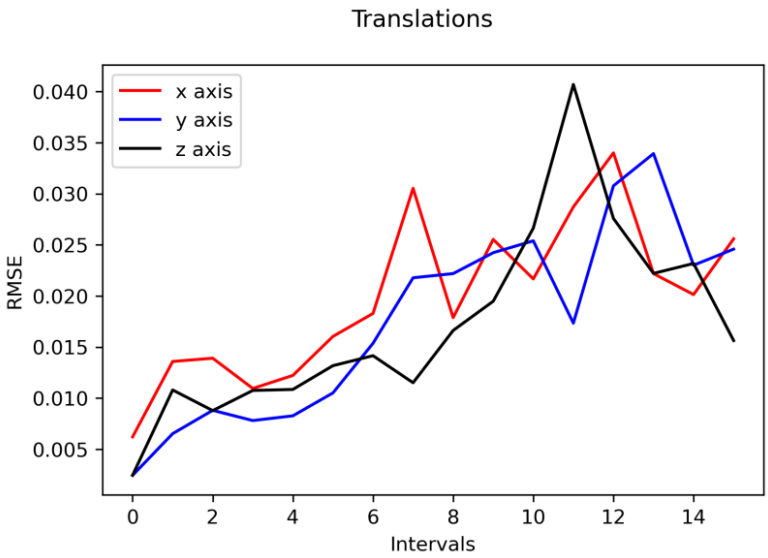

Figure B.1: RMSE values for each step in Neural Network model for translations

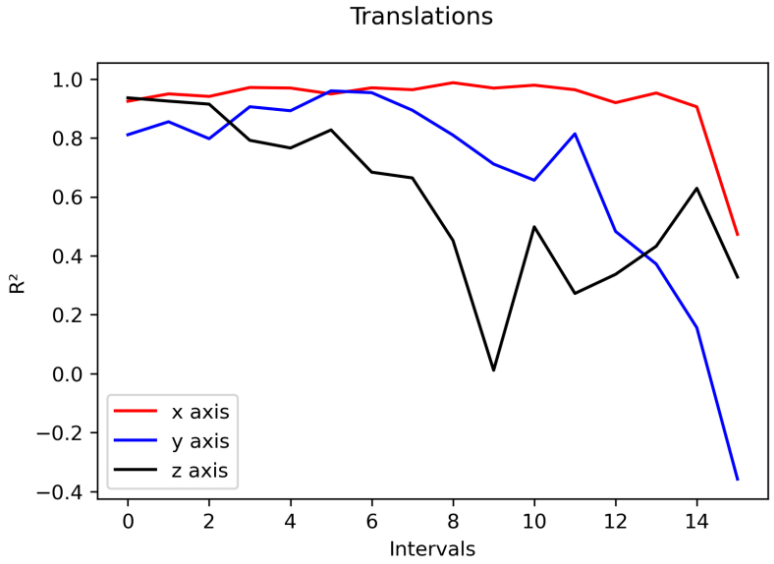

Figure B.2: R^2 values for each step in Neural Network model for translations

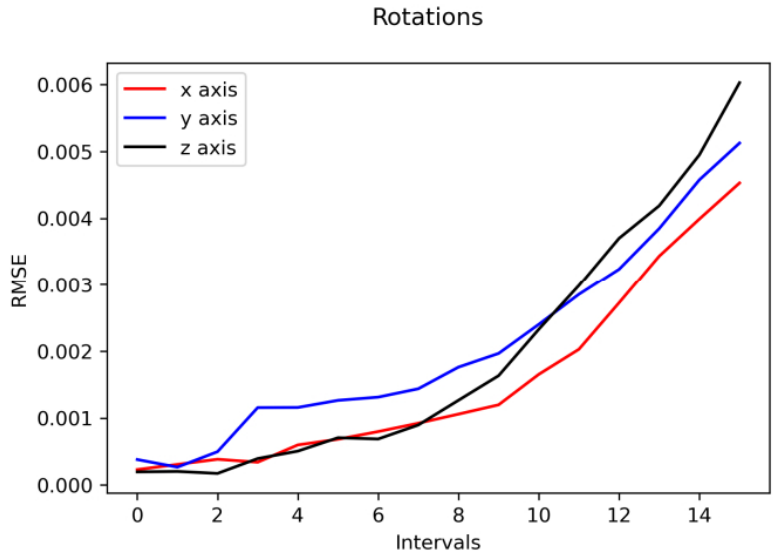

Figure B.3: RMSE values for each step in Neural Network model for rotations

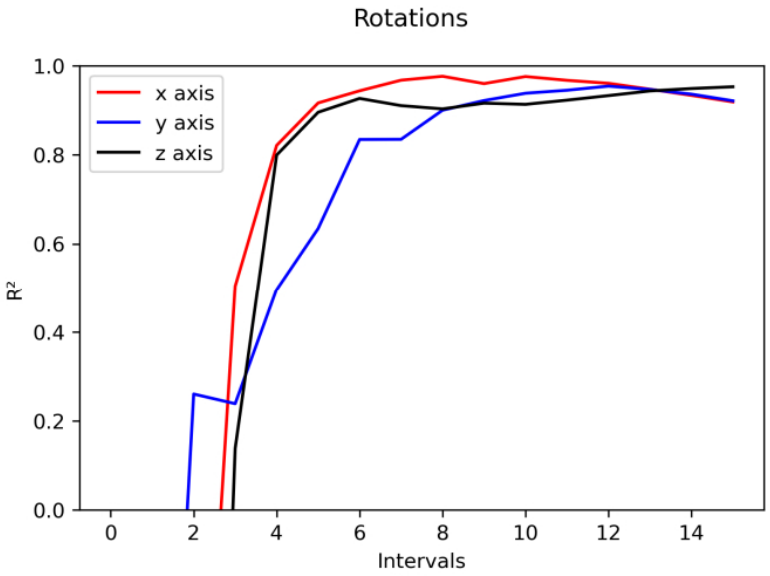

Figure B.4: R^2 values for each step in Neural Network model for rotations

B.2 Support Vector Machine

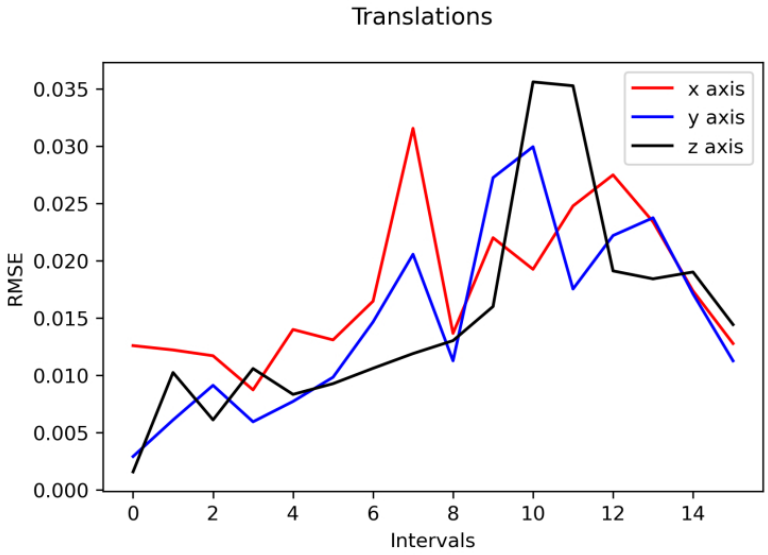

Figure B.5: RMSE values for each step in Support Vector Machine model for translations

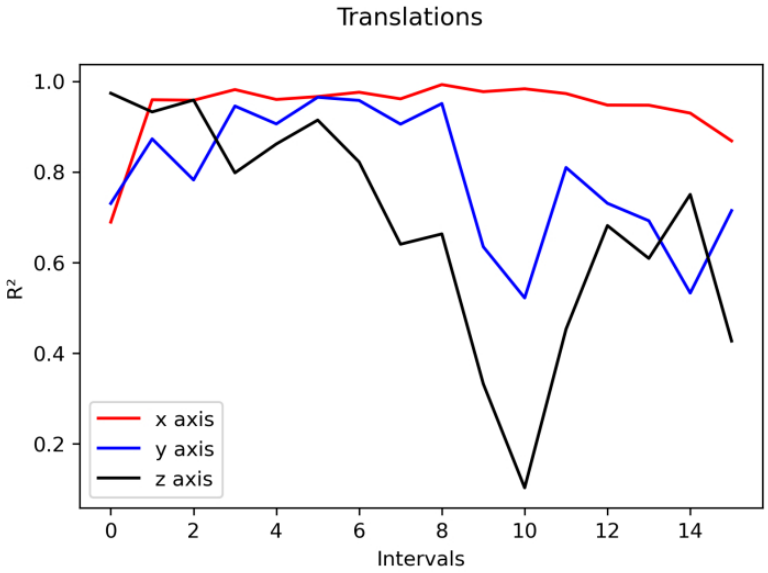

Figure B.6: R^2 values for each step in Support Vector Machine model for translations

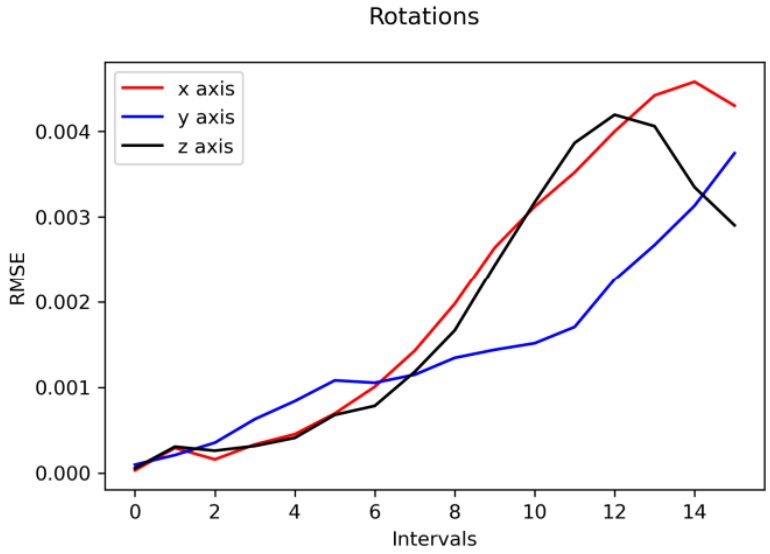

Figure B.7: RMSE values for each step in Support Vector Machine model for rotations

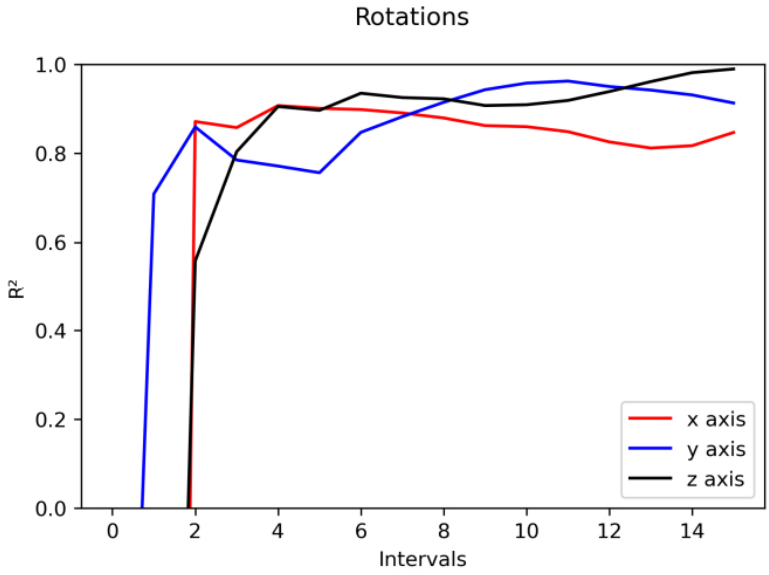

Figure B.8: R^2 values for each step in Support Vector Machine model for rotations

B.3 Gradient Boost

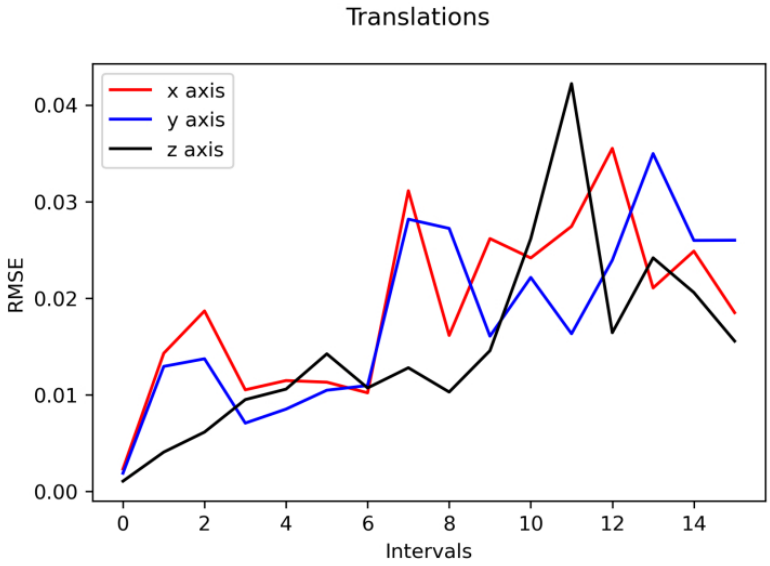

Figure B.9: RMSE values for each step in Gradient Boost model for translations

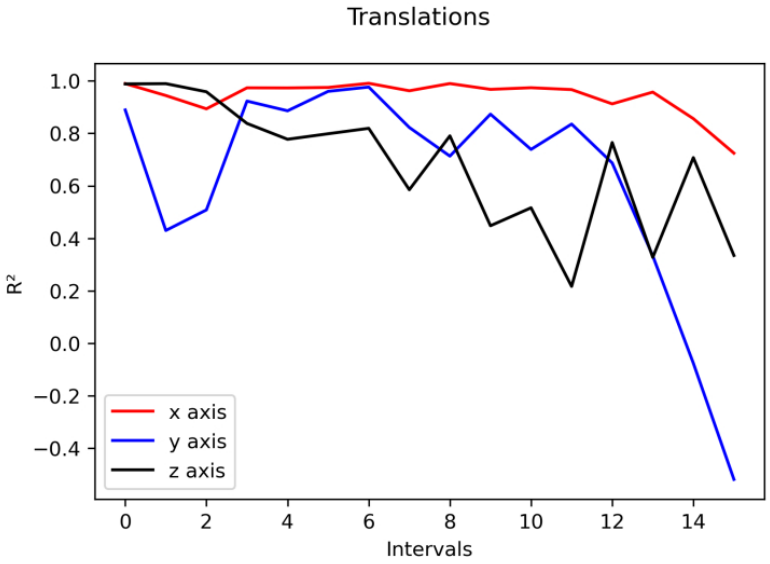

Figure B.10: R^2 values for each step in Gradient Boost model for translations

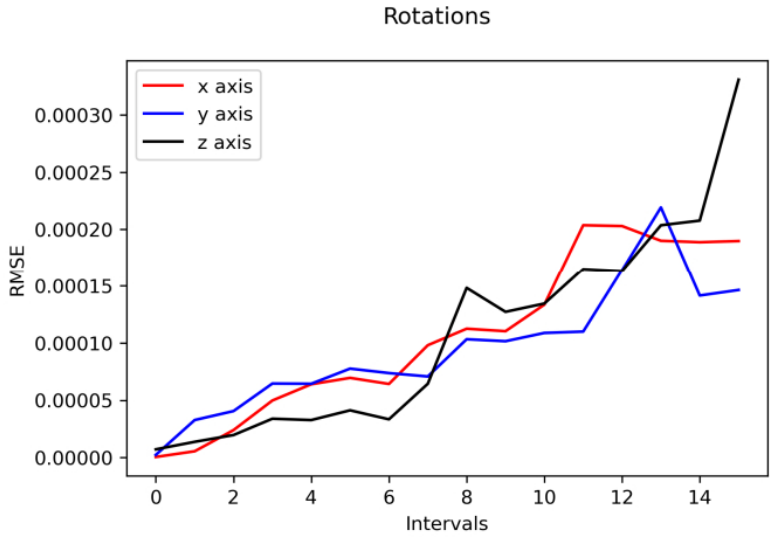

Figure B.11: RMSE values for each step in Gradient Boost model for rotations

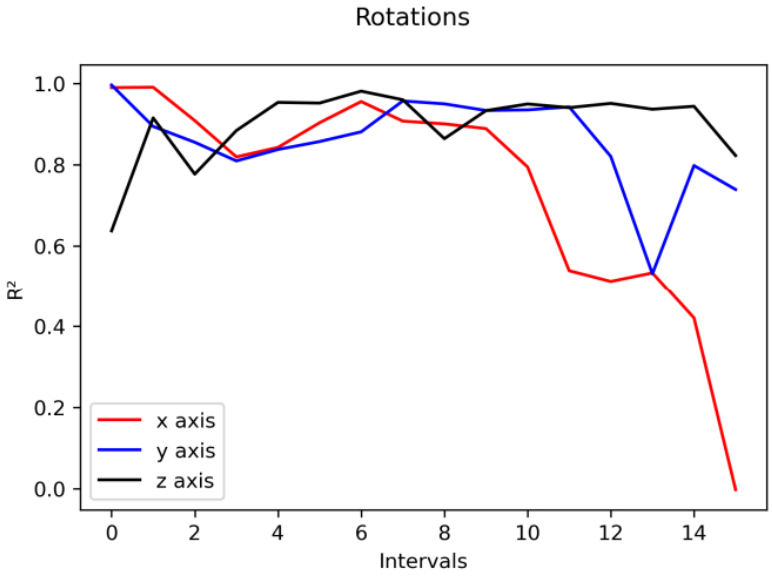

Figure B.12: R^2 values for each step in Gradient Boost model for rotations

B.4 Random Forest

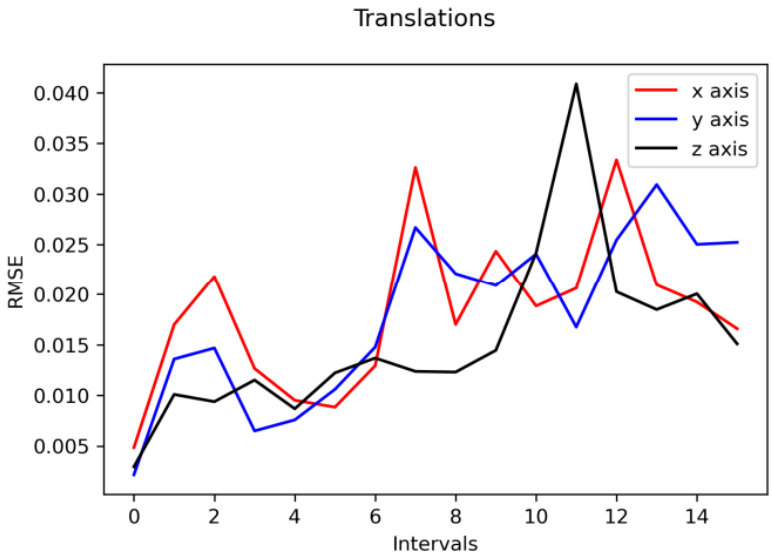

Figure B.13: RMSE values for each step in Random Forest model for translations

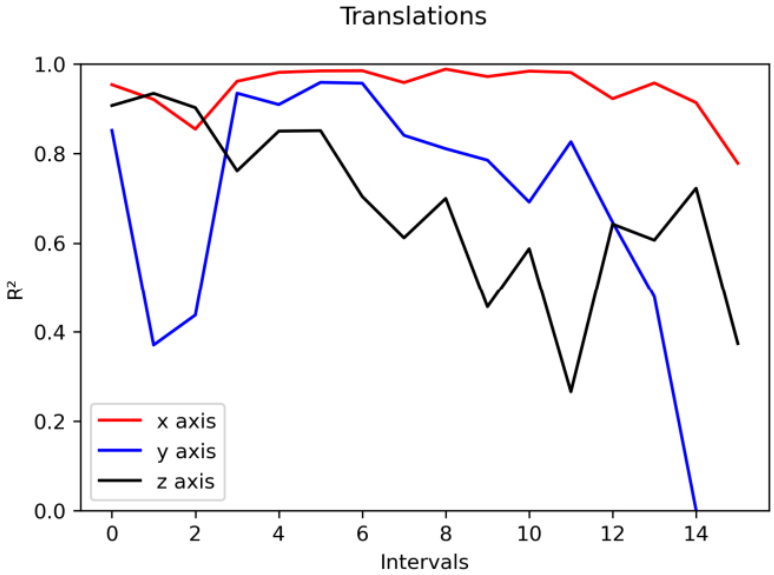

Figure B.14: R^2 values for each step in Random Forest model for translations

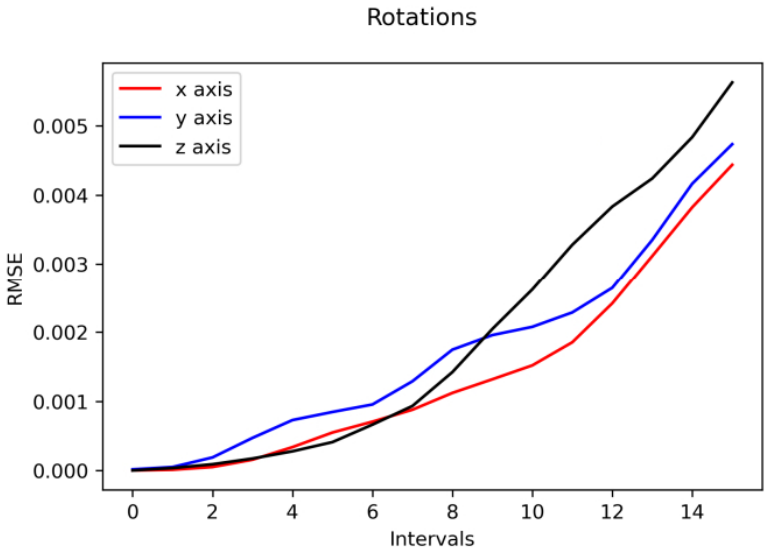

Figure B.15: RMSE values for each step in Random Forest model for rotations

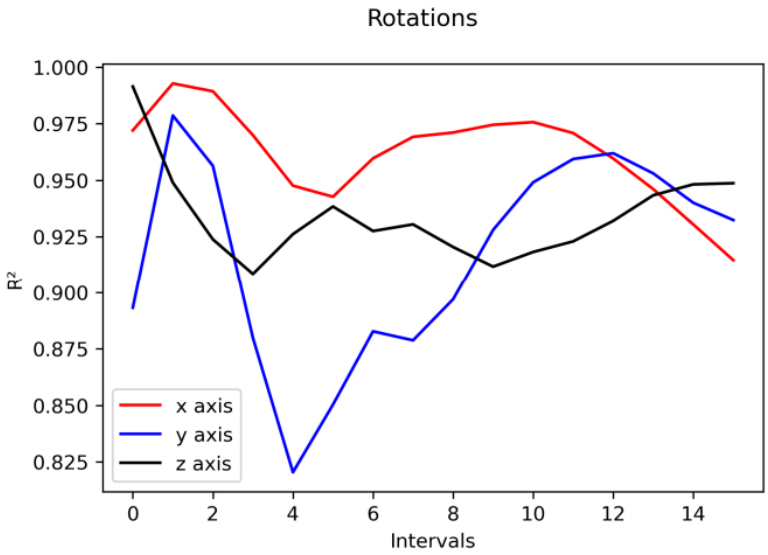

Figure B.16: R^2 values for each step in Random Forest model for rotations