UNIVERSITÀ DEGLI STUDI DI PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

Master degree in ICT for Internet and Multimedia

Recognition of Road Irregularities by means of Low-Cost Sensors and Machine Learning Techniques

Supervisor
Professor Andrea Zanella

Master Candidate
Mattia Pasti

ID Number
2055738

Academic Year
2022–2023

To my family and friends.

Abstract

The work of this thesis aims at automatically recognize road irregularities (potholes, bumps, etc.) with the aid of a variety of low-cost sensors mounted in soft-mobility means (i.e., electric kick-scooters and electric bikes) and machine learning techniques. The work can be divided into three main steps: data acquisition, signal processing and training and performance assessment. In the first part, low-cost sensors (sonars, cheap cameras, accelerometers) are mounted in the vehicle and connected to an Arduino MKR1010 Wi-Fi board that sends the signals to another board connected to the computer through a Wi-Fi connection. Several runs with different road conditions have been taken into account and the signals have been labeled accordingly. The dataset collected during the first step is preprocessed, filtered and then given in input to a variety of machine learning classifiers in order to train the model to automatically recognize different road irregularities. Once the classifiers are trained, the model is tested in a real testbed and the performances of all the algorithms taken into account evaluated, finally selecting the most suitable algorithm.

Contents

Listing of figures

Listing of tables

1 Introduction

A road user may be defined vulnerable with regard to their degree of protection in traffic, such as pedestrians, cyclists, non-motorised road user and motorcyclists, or with regard to their degree of mobility, such as the young, the elderly, and people with disabilities or special needs. A study published in the Journal of Safety Research [1] in 2019 stated that, based on the traffic accident statistics, in the whole European Union, VRUs constitute 46% of all traffic fatalities and 53% of all serious injured accident victims. This problem is expected to be further exacerbated by the quick diffusion of electric kick-scooter and bicycle sharing services, which have markedly increased in popularity in the recent years throughout Europe and the rest of the globe, significantly raising the number of VRUs. Since this expansion has not brought along an improvement in safety, several cities are now questioning the appropriateness of these services, and some (such as Paris) have banned electric kick-scooters sharing services.

This thesis aims to set the ground for a fully functional driver assistance to improve the safety of VRUs by means of sensing systems with which to equip soft mobility vehicles (mainly, electric kick-scooters and electric bikes). Hence, the final goal is to exploit the sensors mounted in the vehicle in order to create a net of interconnected users that exchange relevant information to improve the safety of the VRUs. To achieve this objective, a few challenges need to be addressed. For instance, the dimension of the vehicles taken into consideration poses challenges in the integration of sizable sensors and their optimal positioning. Additionally, attention must be given to the energy consumption of these sensors. Furthermore, a crucial aspect to consider is which information shall be transmitted to other users and with what frequency. In this context, the concept of "Value of Information" (VoI) becomes pivotal. The VoI can be described as the maximum benefit that can be gained in the process of minimizing the average cost with the help of a given amount of information [2]. In other words, the VoI is not set a priori by the sender, but is determined by considering the different utility that a certain information may have for the recipient, based on its context awareness. The issue of information exchange on roadways has been extensively

addressed in the current literature, particularly with the advancement of autonomous driving research. However, the inclusion of VRUs introduces an additional layer of complexity, because VRUs do not strictly adhere to predefined traffic rules during their movements, thus posing another challenge in information exchange.

1.1 Related work

Over the past few years, numerous research papers have been published with the objective of improving the safety of VRUs. The research can be divided into two main categories: collaborative and standalone approaches. The state-of-the-art of both approaches is described in the following of this section.

1.1.1 Collaborative approaches

This approach seeks to establish an effective collaborative environment among all road users and road stations, facilitating the exchange of valuable information aimed at mitigating the risk of accidents. Merdrignac et al. [3], [4] proposed a synergistic integration of Vehicle-to-Pedestrian (V2P) and Vehicle-to-Vehicle (V2V) communication systems, coupled with perception systems utilizing onboard sensors within the vehicle. This innovative approach aims to mitigate the limitations inherent in each individual method and, in turn, enhance the safety of road users. The authors explained that the sensors integrated into vehicles are contingent upon factors such as visibility conditions and inherent imprecisions. Conversely, V2P communication is significantly influenced by the dependability of the information exchanged, which primarily comprises Global Positioning System (GPS) data. Through their research, they demonstrated that the amalgamation of these two distinct methodologies generates a level of redundancy that serves to overcome the shortcomings inherent in each method when employed in isolation. Consequently, this integrated approach contributes substantively to the overall enhancement of safety for VRUs. The development of V2V safety applications based on Dedicated Short Range Communication (DSRC) has been extensively undergoing standardization for more than a decade. Amin Tahmasbi-Sarvestani et. al. [5] introduced an end-to-end V2P framework aimed at enhancing situational awareness and hazard detection in the context of the most prevalent and injury-prone crash scenarios. In their research, Parag Sewalkar and Seitz Jochen [5] observed that V2P systems are customized to distinct goals, primarily focused on improving safety by addressing the needs of different groups of VRUs. Also, these V2P systems integrate a range of communication technologies and use different mechanisms to effectively engage the different groups of users. An effective V2P system must take into account the different characteristics of different VRUs as these multifaceted components comprise critical design parameters within the overall system. Accordingly, Sewalkar and Jochen proposed a comprehensive design framework for V2P systems based on these considerations. Yina Wu et. al. [6] developed a crash warning system for the bicycle lane area at intersection investigating drivers' right turn behavior based on the trajectory data, and they proposed a vehicle-bike crash warning algorithm to calculate the Post Encroachment Time (PET),

that represents the time difference between a vehicle leaving the area of encroachment and a conflicting vehicle entering the same area.

1.1.2 Standalone approach

This type of approach aims to mainly monitor the road conditions in order to improve users' safety. Existing methods for monitoring road conditions are mainly based on camera observation and/or vibration detection.

Vision-based approaches

The vision-based approaches use images to detect the presence of irregularities in the road pavement through computer vision algorithms. Anas Al-Shaghouri et. al. [7] deployed and tested different deep learning architectures to detect the presence of potholes within roadways. The operational methodology involved capturing multiple images of potholes with a smartphone mounted in the windshield of the vehicle, then they tested and compared the performances of different real-time deep learning algorithms with different configurations including TensorFlow, YOLOv3-Darknet53 and YOLOv4CSPDarknet53. Employing the specified configuration, YOLOv4 demonstrated an accuracy of 85% in effectively identifying potholes. Muhammad Haroon Asad et. al. [8] explored the potential of deep learning technique exploiting the Artificial Intelligence (AI) kit on a Raspberry Pi as an edge platform for pothole detection. Hence, they presented a detailed real-time comparison of state-of-the-art deep learning models for pothole detection. As a result, they confirmed that YOLOv4 and YOLOv5 show the highest mean average precision of above 80%. Abhishek Kumar et. al. [9], instead, based their research in the implementation of Convolutional Neural Networks in order to detect potholes using images and videos. They also demonstrated how this type of approach outperforms already existing methods.

Vibration approaches

Once again, many methods have been used, which can be divided into these three categories:

- **Threshold-based methods**
  Threshold based approaches identify anomalies when the amplitude or other properties of the signal exceed a specific value. Prashanth Mohan and colleagues [10] proposed a system to find potholes and bumps utilizing two detectors depending on the speed of the vehicle. After calibration to have consistent reference systems, a spike along the a_z coordinate above a certain threshold is classified as a suspected anomaly. Over several observations, the detector maintained a low false positive rate $(5-10\%)$, but a higher false negative rate $(20-30\%)$. Naveed Sabir et. al. [11] described a road monitoring system using dynamic threshold and crowdsourcing approaches to locate potholes and speed breakers. In this case, a potential pothole is detected when the acceleration values on the x (lateral) and z (vertical) axes exceed the specific threshold, and a speed bump is detected whenever the acceleration values of the y (longitudinal) and z axes exceed the specific threshold. Despite their simplicity, the

threshold based methods achieve fairly good results (90% successful speed bump detection and 85% successful potholes detection).

- **Dynamic Time Warping (DTW)**
  DTW is a pattern-matching algorithm for measuring the similarity between two temporal sequences, which may vary in time and space [12]. For example, Singh [13] proposed a DTW-based approach to detect road surface anomalies from accelerometer sensor data. In this approach, time series values captured accelerometer sensor data for every pothole and bump, and the data were then stored in a central server as templates. Next, incoming sensor data were compared with the stored templates to detect similarities. The accuracy of this approach was greatly correlated to the quality of the reference template. Therefore, this approach was both computationally intensive and unreliable, as it required reference templates for each different condition (i.e., various vehicles, road conditions, speed of driving).

- **Machine Learning together with Feature Engineering**
  Machine Learning techniques have been widely adopted in road inspection, reaching a correct classification rate of above 95%. Chao Wu et. al. [14] analyzed different multiclass machine learning techniques to effectively classify road surface conditions using accelerometer and GPS data collected from a smartphone. In particular, they evaluated the performances of a linear, an SVM and a Random Forest classifier. All these algorithms exhibited good classification performances for both normal road segment and potholes. Akanksh Basavaraju et. al. [15] conducted a comprehensive analysis of various multiclass machine learning approaches to effectively classify road conditions using data collected from smartphone. Their study emphasized the enhanced accuracy achieved by incorporating features from all three axes of the accelerometer, in contrast to relying on a singular axis. Moreover, the research assessed the effectiveness of deep neural networks in road condition classification, achieving an accuracy exceeding 90% without the need for explicit manual feature extraction. Mikko Perttunen and colleagues [16] performed a spectral analysis of tri-axis acceleration signals gathered through the sensor embedded in a mobile phone. They also proposed a speed dependence removal approach for feature extraction and they demonstrated its positive effect in multiple feature sets for the road surface anomaly detection task. Lastly, they proposed a framework for visually analyze the SVM classifier prediction over the validation data and labels. Kshitij Pawar et. al. [17], focused on building an efficient pothole detection system using a Neural Network, while also ensuring that the process is not too complex and time-consuming. Their proposed architecture comprises a data collection stage, followed by data preprocessing and training of the proposed model, ending with the evaluation of the machine learning classifier, which achieved an accuracy of 94.78%.

Previous research mostly focuses on the use of smartphone's built-in sensors, such as accelerometer, gyroscope, GPS, and Wi-Fi signals, to collect data for the detection of potholes or irregularities on road pavement. The use of such devices, however, can easily raise some problems, related to, e.g., the security and privacy of the data communicated to other users, the assumption that the smartphone is always connected and online, the heterogeneity of device capability and so on.

Furthermore, utilizing the user's smartphone also necessitates the development of a custom application that users must download and connect to their vehicle to manage data from these sensors. This additional step not only requires user downloads but also increases the overall application costs, which is contrary to the objective of maintaining a low-cost solution. As such, in this project we propose to mainly leverage onboard sensors and processing, so as to guarantee a certain level of safety to each vehicle without needing any intervention of the user.

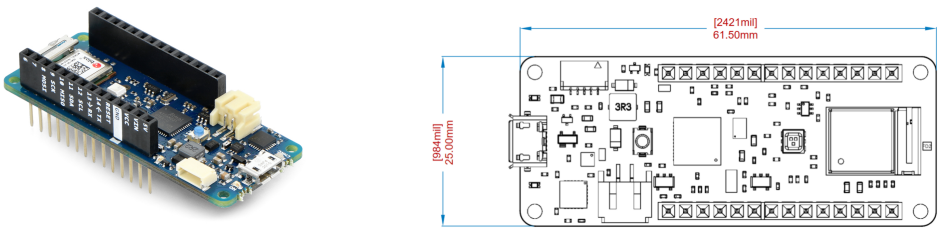
2 Methodology

2.1 Hardware

As outlined in the introduction, the use of low-cost sensors to detect road anomalies is pivotal. The following subsections aim to present the hardware utilized for this research.

2.1.1 Arduino MKR WiFi 1010

The Arduino MKR WiFi 1010 is the easiest point of entry to basic IoT application design. The board's main processor is a low power Arm® Cortex®-M0 32-bit SAMD21. The WiFi and Bluetooth® connectivity is performed with a module from u-blox, the NINA-W10, a low power chipset operating in the 2.4GHz range. On top of those, secure communication is ensured through the Microchip® ECC508 crypto chip [18].



(a) *Board image.* **(b)** *Board scheme.*

Figure 2.1: Arduino MKR WiFi 1010 board.

In our study, this type of Arduino board is employed for the purpose of gathering data from the sensors mounted in the vehicle, which is subsequently transmitted to a separate Arduino MKR WiFi 1010 unit. This latter unit, in direct connection with the computer, is responsible for the

reception, reading, and storage of the acquired data signals. In this early stage of the application, the utilization of two separate boards is advantageous for the more convenient gathering of data, since it is not necessary to connect the computer directly to the board embedded in the bicycle. The board implements different types of communication protocol useful for the project that are listed below:

- **Wi-Fi protocol**: exploiting Arduino's WiFiNINA library [19], the Wi-Fi module enables the Arduino MKR Wi-Fi 1010 to function both as a server, accepting incoming connections, and as a client, establishing outgoing ones. This configuration allows the establishment of a UDP connection between the two boards, facilitating the seamless transmission of collected data from the board embedded into the vehicle to the other board connected to the computer. Notably, this wireless configuration eliminates the need for a physical wired connection between the vehicle board and the computer, easing the data collection process.

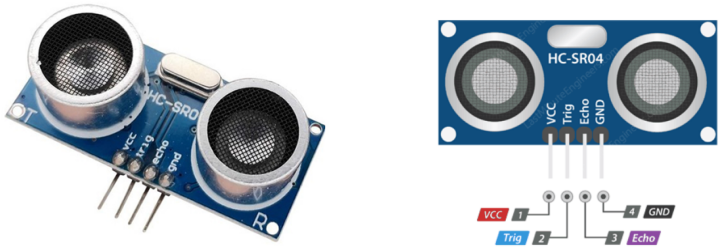
- **UART protocol**: UART, an acronym for Universal Asynchronous Receiver/Transmitter, defines a protocol or set of rules governing the exchange of serial data between two devices. It operates using a simple mechanism employing only two wires to transmit and receive data bidirectionally. Communication in UART can take three forms: simplex (data is sent in one direction only), half-duplex (each side speaks but only one at a time), or full-duplex (both sides can transmit simultaneously). A significant advantage of UART lies in its asynchronous nature, where the transmitter and receiver do not rely on a common clock signal. While this simplifies the protocol, it necessitates that both ends transmit at a pre-arranged, agreed-upon speed to maintain consistent bit timing. The most prevalent UART baud rates in contemporary usage include 4800, 9600, 19.2K, 57.6K, and 115.2K [20].

- **I2C protocol**: the Inter-Integrated Circuit (I2C) communication protocol is structured on the master-slave paradigm. Similar to UART, it utilizes a two-wire system for communication, namely the Serial Data (SDA) and Serial Clock (SCL) wires. However, unlike UART, I2C is synchronous in nature, meaning the output of bits is synchronized to the sampling of bits through a clock signal shared between the master and the slave. The clock signal is always controlled by the master. In the I2C protocol, data is transferred in messages, and messages are further broken down into frames of data. Each message comprises an address frame containing the binary address of the slave, along with one or more data frames containing the transmitted data. In an I2C communication, the master initiates by sending the address of the slave it wishes to communicate with to all slaves connected to it. Each slave then compares the address sent by the master with its own address. If a match is found, the slave responds with a low voltage acknowledgment (ACK) bit to the master. Conversely, if there's no match, the slave remains inactive, and the SDA line retains a high voltage [21].

2.1.2 HC-SR04 Ultrasonic Ranging Module

The HC-SR04 Ultrasonic Ranging Module exploit ultrasonic transmitters, receiver and control circuit to give an estimate of the distance between the module and an object placed in front of it.

It measures distance between 2cm and 400cm, with a ranging accuracy that can reach 3mm.



(a) *Sonar sensor image.* **(b)** *Sonar sensor scheme.*

Figure 2.2: HC-SR04 Ultrasonic Ranging Module.

The schematic representation, depicted in Fig. 2.2(b), illustrates the four pins through which the sensor connects to the board for collecting necessary data. These pins include the input voltage pin ($VCC = 5\text{V}$), the ground pin (GND), as well as the trig and echo pins responsible for transmitting and receiving the signal to estimate the distance of the object that caused the reflection. The sonar sensor operates based on the following principles:

- Exploit the IO trigger to send a 10μs high level signal
- The module automatically sends eight-cycle burst of ultrasound at 40KHz and raise its echo
- Estimate the distance of the object that generates the echo signal with the formula

$$d = \frac{ToF \times v}{2} \tag{2.1}$$

where ToF is the total time of flight of the signal (time it takes to reach the object plus the time it takes to return), v is the velocity of the sound ($\simeq 340\text{m/s}$), all divided by 2 since the signal's travel path covers a distance equivalent to twice the separation between the sonar sensor and the object of interest.

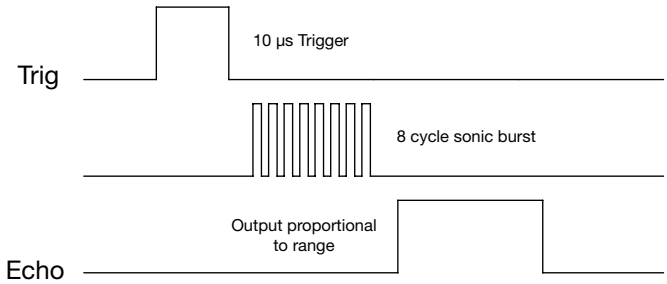


Figure 2.3: Sonar functioning.

In Fig. 2.3 above, a basic schematic is presented to elucidate its functioning. Initially, a high signal is transmitted, triggering the generation of an eight-cycle burst of ultrasound at a frequency of

40KHz. The duration of the high Echo Back signal corresponds to an estimate of the *ToF* of the reflected ultrasound signal. This *ToF* is functional to accurately determining the distance of the object that generated the echo.

2.1.3 MPU-6050 GY-521 Accelerometer-Gyro Module

The MPU-6050 sensor module combines a three-axis gyroscope, three-axis accelerometer, and a Digital Motion Processor (DMP) all in a small $4 \times 4 \times 0.9$mm package.

(a) *Accelerometer Image.* **(b)** *Accelerometer scheme.*

Figure 2.4: MPU-6050 GY-521 Accelerometer-Gyro Module.

The multiple features integrated in the module are explained below:

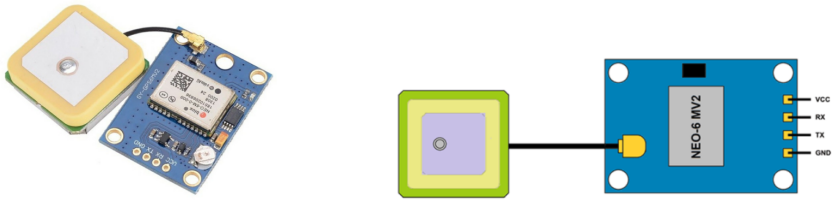
- **Three-axes accelerometer**: can detect the tilt angle or inclination along the x, y, z axes using a Micro Electro-Mechanical Systems (MEMS) technology. The acceleration along the axes is derived from a moving mass and the amplitude of the output is proportional to the acceleration undergone by the mass. A 16-bit ADC is used to digitize the values. The possible full-scale output ranges are $\pm$2g, $\pm$4g, $\pm$8g, $\pm$16g depending on the the sensitivity settings chosen. In the normal position, that is, when the device is placed on a flat surface, the values are 0g on the x-axis, 0g on the y-axis, and $+1$g on the z-axis.

- **Three-axis gyroscope**: can detect the rotation speed along the x, y, z axis with Micro Electro-Mechanical System (MEMS) technology. When the sensor is rotated on any axis, a vibration is produced and due to the Coriolis effect is detected by MEMS. The full-scale output ranges are : $\pm$250, $\pm$500, $\pm$1000, $\pm$2000. Angular velocity is measured along each axis in degrees per second units.

In addition to these components used in the project, this module implement also a temperature sensor, an internal clock oscillator and a digital motion processor. As shown in the Fig. 2.4(b), the accelerometer module comprise eight pins, however, only four of them are used in this project, namely the input voltage pin (3 to 5V), the ground pin and the serial clock and serial data pins to propagate the information towards the board. The communication between the board and the sensors is facilitated through the I2C protocol explained in the previous subsection.

2.1.4 GY-NEO6MV2 GPS Module

GY-NEO6MV2 module series is a family of stand-alone GPS receivers featuring the high performance U-Blox 6 positioning engine. The module simply checks its location on earth and provides output data which is the longitude and latitude of its position.



(a) *GPS Module Image.* **(b)** *GPS Module scheme.*

Figure 2.5: GY-NEO6MV2 GPS Module.

To offer better signal reception, there is an external ceramic antenna that connects to the board via a U.FL connector. The compact architecture, modular design, and efficient power and memory options of NEO6MV2 modules render them highly suitable for battery-operated mobile devices constrained by stringent cost and space requirements. This makes such components a perfect fit for the project described in this thesis. The four pins shown in Fig. 2.5(b) make the connection with the board very simple. The *VCC* and *GND* pins are connected to the 3.3V and *GND* pins of the board, respectively, and the *tx* and *rx* pins to the *rx* and *tx* pins of the board. Differently from the sonar sensor described in the previous section, the communication follows the UART protocol.

2.1.5 External battery

To power the Arduino board mounted in the vehicle, and consequently all the sensors connected to it, an external battery of 2200mAh capacity and 5 output Volt has been used.

2.2 Final scheme

All the modules described in Section 2.1 have been embedded in the bicycle used for data acquisition. Fig. 2.6 shows a picture of the considered vehicle.
A breadboard was secured to the bike frame. Subsequently, the Arduino board was mounted onto the breadboard along with the accelerometer module. Following this, a series of wires were employed to establish connections between the sonar sensor, positioned in the lower part of the bicycle, and the board. Placing the accelerometer in a stable position on the bicycle frame was crucial to ensure a steady and reliable data acquisition. Therefore, careful consideration was given to choose the most accurate location on the bicycle frame for optimal stability. Similarly, the positioning of the sonar sensor was a strategic choice. Mounting it under the pedal was determined

by the objective of maintaining stability, ensure that the sonar was perpendicular to the road surface and secure that it was at the center of vehicle in both horizontal directions. This arrangement aimed to have the signal transmitted and received by the sonar at the desired angle of incidence to maximize the efficiency and accuracy of pothole and bumps classification.

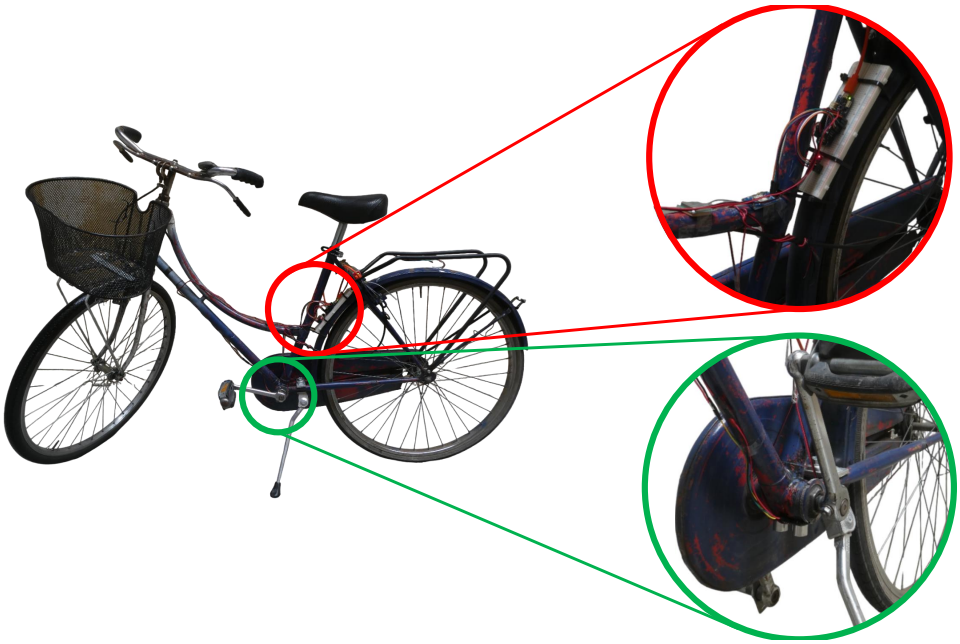


Figure 2.6: Photo of the vehicle considered for the data acquisition and details of the embedding of the sensors.

3

Machine Learning Techniques

In this chapter, a comprehensive theoretical exposition is provided on machine learning techniques used to classify road conditions. The adoption of machine learning in this context is underpinned by its inherent versatility and its adaptability to diverse datasets and complex patterns inherent in pavement conditions. Simple threshold based methods should be continually refined in order to conduct a proper classification when the conditions are changing. The capability of machine learning algorithms to automatically learn and adapt from data, without explicit programming for each condition, renders them highly suitable for the variable nature of road pavement classification. While the landscape of machine learning classifiers ranges from simpler to more complex models, the selection of the algorithms for this thesis prioritizes memory and computation efficiency. This deliberate choice leans towards a simpler yet accurate algorithm, aligning with the objective of maintaining low "costs" while ensuring effective classification for road pavement conditions. Therefore, the algorithmic selection in this context favors Support Vector Machines, Random Forest, and K-Means classifiers.

It's important to highlight that when training a machine learning classifier, the risk of overfitting or underfitting can arise. Underfitting occurs when a statistical model or machine learning algorithm is too simplistic to capture the complexities present in the data. It represents the inability of the model to learn the training data effectively, resulting in poor performance both on the training and testing data. In simple terms, an underfit model produces inaccurate predictions, particularly when applied to new, unseen examples. It mainly happens when we uses very simple model with overly simplified assumptions. Conversely, overfitting happens when a statistical model fails to make accurate predictions on testing data. This occurs when a model is trained with an excessive amount of data, causing it to learn from noise and inaccuracies in the dataset. Consequently, when tested on new data, the model exhibits high variance and struggles to categorize the data accurately due to an excess of details and noise learned during training. To address these challenges, the dataset is divided into three distinct sets: *training*, *validation*, and *test* sets.

- Training Data: this set is used for the actual training of the model, allowing it to learn from the provided data.
- Validation Split: the validation set plays a crucial role in refining and enhancing the model's performance. It is employed for fine-tuning the model after each training epoch, aiding in optimizing its parameters and improving generalization.
- Test Set: the test set is utilized to evaluate the final accuracy of the model after completing the training phase. It provides insights into how well the model performs on new, unseen data.

In situations where the dataset size is insufficient to create three separate sets, the *K-fold cross-validation* methodology is employed. This technique involves partitioning the dataset into K subsets (folds), using $K-1$ folds for training and the remaining one for validation. The process is repeated K times, each time using a different fold for validation, and the results are averaged. This helps in obtaining a more robust evaluation of the model's performance, especially when data availability is limited.

3.1 Supervised Learning

Supervised learning techniques represent the most prevalent methods in machine learning. These approaches utilize a training set to instruct models on how to produce the desired output. The training dataset consists of inputs paired with correct outputs, allowing the model to learn patterns over time. The algorithm evaluates its accuracy through a loss function, iteratively adjusting its parameters until the error has been sufficiently minimized. Supervised learning can be categorized into two primary types, i.e., classification and regression.

- Classification: employs an algorithm to accurately categorize test data into specific classes or categories. It identifies distinct entities within the dataset and aims to make informed decisions about how to label or define those entities. Common classification algorithms include Support Vector Machines, decision trees and Random Forest, described in detail in the next subsections.
- Regression: utilized to comprehend the relationship between dependent and independent variables. It is commonly applied for making predictions or projections, such as estimating sales revenue for a given business. Popular regression algorithms encompass linear regression, logistic regression, and polynomial regression.

In the following of this section, we will discuss in greater detail some of the most common supervised learning techniques for classification and regression, namely Support Vector Machine and Random Forest in the context of supervised learning, and K-Means clustering technique concerning the unsupervised learning.

3.1.1 Support Vector Machines (SVM)

SVM is one of the most common supervised machine learning techniques used for classification and regression problems. The idea of SVM is fairly simple: the algorithm creates a line or a hyperplane which separated the data into classes. In order to be able to fully understand the reasoning behind this type of classifier, a few concept must be explained. Suppose to have a classification problem with two classes and assume the data to be linearly separable. We can hence define the *margin* as the minimum distance from an example in the training set to the hyperplane. There are different methods for initializing the hyperplane, and the choice of the initialization can impact the convergence speed and the quality of the final solution. Some common methods include a random initialization, use some data points and set the hyperplane to be in the middle on the selected instances, use a previous model parameters (if possible) and others. The idea is that the best final separating hyperplane is the one that maximizes the margin. Given two vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ and given a bias term b, we can define the separating hyperplane as:

$$L = \{\boldsymbol{v} :< \boldsymbol{v}, \boldsymbol{w} > +b = 0\} \tag{3.1}$$

where the symbol $< \bullet, \bullet >$ represents the dot product between two vectors. The vector $\boldsymbol{v}$ denotes a point within the feature space and the vector $\boldsymbol{w}$ serves as the weight vector linked to the features of the data. These weights signify the significance of each feature in the classification process and collectively determine the orientation of the hyperplane. The bias term b is essentially an offset that allows the SVM to shift the decision boundary away from the origin. It represents the intercept of the equation. The decision rule is based on the sign of the expression: if $< \boldsymbol{v}, \boldsymbol{w} > +b < 0$ the point is classified as one class, and if $< \boldsymbol{v}, \boldsymbol{w} > +b > 0$, then it is classified as the other class. The distance of an example in the training set $\boldsymbol{x}$ to L is:

$$d(\boldsymbol{x}, L) = \min\{\|\boldsymbol{x} - \boldsymbol{v}\| : \boldsymbol{v} \in L\} \tag{3.2}$$

where the symbol $\| \bullet \|$ represent the norm of the vector. It can be proved that, if $\|\boldsymbol{w}\| = 1$, then the distance between a sample and the hyperplane is $d(\boldsymbol{x}, L) = | < \boldsymbol{w}, \boldsymbol{x} > +b|$, where the symbol $| \bullet |$ stands as the absolute value of quantity enclosed within. In this case, the margin is:

$$\min_{\boldsymbol{x}_i \in \mathcal{S}} | < \boldsymbol{w}, \boldsymbol{x}_i > +b| \tag{3.3}$$

where $\mathcal{S}$ is the training dataset. The closest samples to the hyperplane are called *support vectors*. SVM techniques can be further divided into two subclasses, i.e., hard-SVM and soft-SVM, depending on the goal to be achieved and the type of dataset.

Hard-SVM

The goal of hard-SVM is to find the separating hyperplane with the largest margin, i.e., find:

$$\underset{(\boldsymbol{w},b):\|\boldsymbol{w}\|=1}{\operatorname{argmax}} \min_i |<\boldsymbol{w},\boldsymbol{x}_i>+b| \qquad \text{subject to} \qquad \forall i : y_i(<\boldsymbol{w},\boldsymbol{x}_i>+b)>0 \tag{3.4}$$

where y_i can be either 1 or -1 and represents the correct classification of the sample $\boldsymbol{x}_i$ (recall that is a supervised learning technique, so we have the ground truth for each of the training instances). It is straightforward that the condition is satisfied when the data is correctly classified. This condition implies that the data has to be linearly separable. In this case, the goal can be written as:

$$\underset{(\boldsymbol{w},b):\|\boldsymbol{w}\|=1}{\operatorname{argmax}} \min_i y_i(<\boldsymbol{w},\boldsymbol{x}_i>+b). \tag{3.5}$$

Soft-SVM

The key problem of hard-SVM is that it needs the data to be linearly separable, which happens rarely in practice. Soft-SVM addresses this limitation by relaxing the constraint and considering instances where separation is not perfect. This is achieved through the introduction of *slack variables*, denoted as $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_m)$, where $\xi_i \geq 0$ for each i. The *slack variables* are designed to quantify and account for the degree of violation of the separation constraint in the optimization process. Now the condition become:

$$\forall i = 1, \ldots, m : y_i(<\boldsymbol{w},\boldsymbol{x}_i>+b) \geq 1-\xi_i \tag{3.6}$$

where ξ_i store how much the constraint is violated. Hence, soft-SVM aims to jointly minimize:

1. the norm of $\boldsymbol{w}$ ($\rightarrow$ maximize the margin);
2. the average of ξ_i ($\rightarrow$ minimize constraint violation).

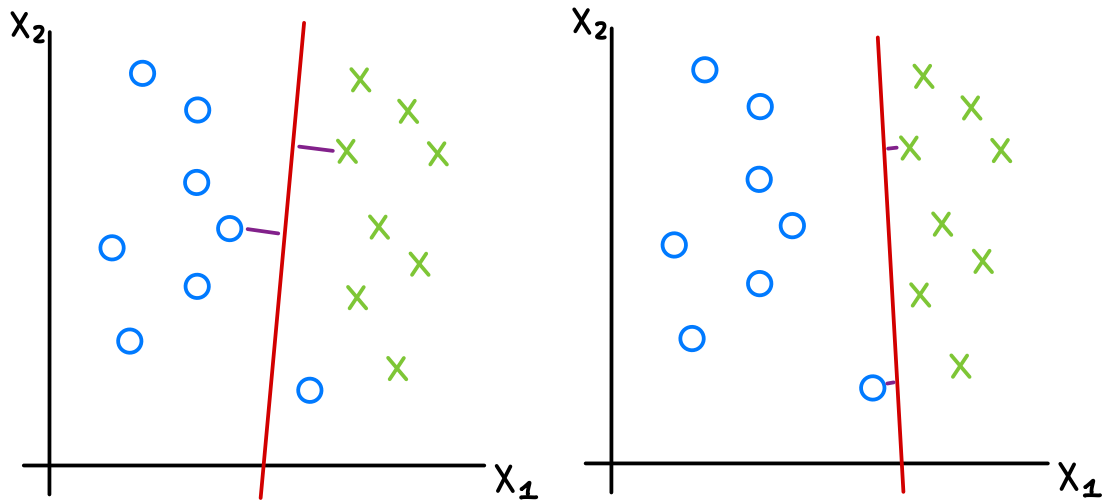
The trade-off between these two objective is controlled by the parameter λ. If the inputs are the couples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$ and the parameter λ, the optimization problem is to solve:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \left(\lambda\|\boldsymbol{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m}\xi_i \right) \qquad \text{subject to} \qquad \forall i : y_i(<\boldsymbol{w},\boldsymbol{x}_i>+b) \geq 1-\xi_i \quad \text{and} \quad \xi_i \geq 0 \tag{3.7}$$

Is straightforward that a large λ means focusing on the margin (if $\lambda \to \infty$ we have hard-SVM) and a small λ means focusing on avoiding errors. An example is showed in Fig. 3.1

3.1.2 Random Forest (RF)

To gain a comprehensive understanding of the RF algorithm, it is essential to grasp the underlying concept of a decision tree. A decision tree serves as a predictive model for associating labels with a given sample vector $\boldsymbol{x}$ by navigating through a tree structure from its root to a leaf node. At



(a) *Large* λ **(b)** *Small* λ

Figure 3.1: Example of soft-SVM with different values of the parameter λ.

each internal node of the tree, a decision is made based on the features present in $\boldsymbol{x}$. This decision-making process corresponds to partitioning the input space, with the simplest approach involving splitting based on a specified threshold applied to one of the input features, i.e., $x_i < \theta$ or $x_i \geq \theta$. Ultimately, each leaf is associated to a label that represents the final prediction. The fundamental idea behind decision trees is relatively straightforward, but the challenge lies in the construction of the tree itself. For instance, in binary classification scenarios and assuming the existence of a performance measure, such as training error, the initial tree might consist of a single leaf node that assigns the most frequent label (i.e., the majority label among the samples). Subsequently, during each iteration, the impact of splitting a leaf node is evaluated. Among all the potential splits, the one leading to the greatest improvement in the performance measure is selected, and the corresponding leaf is split accordingly, or the decision may be not to split further. A crucial issue in this process is the selection of the feature to split on since there are numerous options, considering all possible thresholds and components. To address this challenge, it is necessary to define a suitable performance measure. The simplest approach is to use training error, where the goal is to select the split that results in the most significant reduction in training error. Alternatively, another commonly employed choice is the *gini* index, which serves as a measure of the statistical dispersion within the dataset. This index informs us about whether the data within our split predominantly belongs to a single class or is distributed among various classes. The *gini* index is defined as:

$$G = \sum_{i=1}^{C} p(i)(1 - p(i)) = 1 - \sum_{i=1}^{C} [p(i)]^2 \tag{3.8}$$

where C is the total number of classes (2 for binary classification) and $p(i)$ is the probability of the class i, simply defined as the ratio between the number of sample belonging to class i and the total

number of samples in the considered node. For sake of clarity, if all sample are in the same class, then $G = 0$, while is the sample are uniformly distributed among all the classes, the *gini* index is equal to 0.5. In essence, the gini index serves as an indicator of the purity of the distribution after a split. It provides a smooth and concave upper bound for the training error.

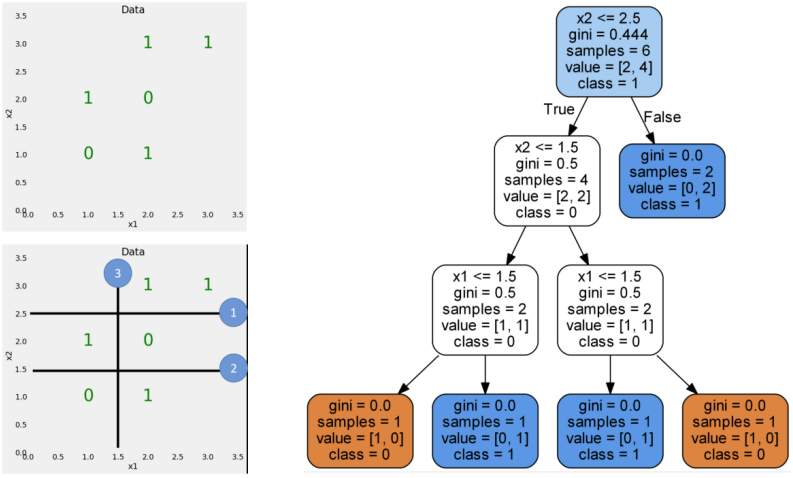


Figure 3.2: Example of a decision tree employed for binary classification.

In the Fig. 3.2 is shown a simple example of the structure of a decision tree employed for binary classification. One can notice that, after the first split, the rightmost node contains samples only belonging to class 1, hence $G = 0$. In contrast, the leftmost node contains four sample in total, two from class 1 and two from class 0. Accordingly, the *gini* index in this specific case is equal to 0.5. In principle, expanding the size of a decision tree ultimately leads to each leaf being associated with a single sample, effectively reducing the *gini* index to zero for all leaves. While this seems good for the prediction, it results in overfitting the data, causing a bad true error when inputting a different sample in the tree.

Research has shown that instead of constructing a single, very large decision tree, it is more advantageous to build multiple smaller trees. A Random Forest (RF) is a classifier composed of a group of decision trees. Predictions in a RF are determined by conducting a majority vote based on the predictions of the individual trees. After the trees are constructed, when presented with an input sample, it is fed into all the trees, and the classification is determined through a majority voting process. This ensemble approach enhances the accuracy and generalization of the classifier. The challenge emerges once again when constructing the decision trees, primarily due to the fact that we have only one training dataset. If we use the same training set for building each of the trees, we end up with a forest consisting of identical trees, rendering the entire process ineffectual. The fundamental concept of a RF is to introduce randomness by applying a process known as "random sampling with replacement" to the dataset. This technique results in the creation of new training datasets that differ from one another, ensuring diversity in the data used for building individual decision trees within the forest. The "random sampling with replacement" technique operates as follows: new datasets with the same dimension as the original training dataset are generated by

randomly selecting items from the original dataset. A key element of this technique is that each sample can be chosen multiple times, or not at all, thereby producing diverse datasets that maintain the same dimension as the original dataset. For example, if the training data is $[1, 2, 3, 4, 5, 6]$, then the resulting sets might be $[1, 2, 2, 5, 6, 6]$, $[2, 3, 3, 4, 4, 6]$, $[1, 1, 1, 1, 1, 1]$ etc. Notice that all lists have length equal to the original training dataset. The technique of generating different decision trees from distinct training datasets is referred to as "bagging", which stands for "bootstrap aggregation". Bagging is highly valuable because decision trees are inherently sensitive to the data they are trained on. Even small variations in the input data can lead to the creation of different trees. RF takes advantage of this by allowing each individual tree to randomly sample with replacement from the dataset, resulting in different training sets producing different trees. The RF algorithm introduces an additional layer of randomness by modifying the way it selects features for node splitting. In a standard decision tree, when a node is to be split, all available features are considered, and the one that maximizes the gain is chosen. In contrast, each tree within a random forest can only select from a random subset of features during node splitting. In other words, the process of splitting nodes in a RF model is grounded in a random subset of features for each tree. This deliberate feature selection strategy heightens the diversity among the trees within the model. It ensures that different trees rely on different subsets of features during their decision-making processes. Consequently, this approach reduces the correlation among the trees, promoting greater diversification and enhancing the overall robustness and predictive accuracy of the Random Forest ensemble.

3.2 Unsupervised Learning

Unsupervised learning employs algorithms to analyze and cluster unlabeled datasets. In this approach, the algorithms autonomously uncover latent patterns or groupings within the data without the requirement for human-provided labels or guidance. Specifically, the focus of our discussion is on clustering techniques. Clustering is the task of grouping a set of objects such that the similar objects end up in the same group and dissimilar object are separated into different groups. In the context of this work, the objective is to differentiate segments of smooth road from sections with irregularities and, ultimately, to classify various types of irregularities based on their severity.
The following subsections provide descriptions of one of the most common clustering algorithm, specifically, K-Means.

3.2.1 Clustering Method – K-Means

The K-Means clustering technique stands as the simplest distance-based clustering algorithm, aiming to identify cluster centers (with the assumption that each cluster has a corresponding center representing it) and allocate data points to clusters. The primary objective is to minimize the approximation error when representing data points with these cluster centers. There are numerous possible allocations for the cluster centers, making infeasible to consider all the possi-

ble combinations. Therefore, K-Means operates as an iterative algorithm that maintains a fixed number of clusters, assigns each data point to the nearest cluster, recalculates the optimal cluster centers, and repeats this process. When utilizing the Euclidean distance as the measure between points and cluster centers, the aim is to minimize:

$$\boldsymbol{\mu}_i(\mathcal{C}_i) = \arg\min_{\boldsymbol{\mu}} \sum_{\boldsymbol{x} \in \mathcal{C}_i} d(\boldsymbol{x}, \boldsymbol{\mu})^2 = \arg\min_{\boldsymbol{\mu}} \sum_{\boldsymbol{x} \in \mathcal{C}_i} \|\boldsymbol{x}, \boldsymbol{\mu}_i\|^2 \tag{3.9}$$

where $\boldsymbol{\mu}_i$ are the centroids of the clusters, $\mathcal{C}_i$ are the clusters and $\boldsymbol{x}$ are the vectors to be clustered. Can be shown that given a cluster $\mathcal{C}_i$, the center $\boldsymbol{\mu}_i$ that minimizes $\sum_{\boldsymbol{x} \in \mathcal{C}_i} d(\boldsymbol{x}, \boldsymbol{\mu})^2$ is the baricenter:

$$\boldsymbol{\mu}_i = \frac{1}{|\mathcal{C}_i|} \sum_{\boldsymbol{x} \in \mathcal{C}_i} \boldsymbol{x} \tag{3.10}$$

The K-Means algorithm works as follows:

1. Select k random centroids
2. Each point is associated to the closest centroid (according to the distance measure taken into account):
$$\forall\, i : \mathcal{C}_i = \{\boldsymbol{x} \in \mathcal{X} : i = \arg\min_j \|\boldsymbol{x} - \boldsymbol{\mu}_j\|\} \tag{3.11}$$
3. Compute the new centroids (each centroid is the baricenter of the associated points):
$$\forall\, i : \boldsymbol{\mu}_i = \frac{1}{|\mathcal{C}_i|} \sum_{\boldsymbol{x} \in \mathcal{C}_i} \boldsymbol{x} \tag{3.12}$$
4. Repeat step 2 and 3 until the algorithm converges

Various stopping criteria exist for this algorithm. These include scenarios where the positions of centroids and point-cluster assignments remain constant, the improvement in error falls below a specified threshold for a set number of consecutive iterations, or the maximum allowable iteration count is reached. This algorithm is fast and simple, with a guaranteed convergence. However, it does not assure an optimal solution (as it relies on the initial centroids), necessitates prior knowledge of the number of clusters (k), and enforces spherical symmetry among clusters.
In the context of this application, the K-Means algorithm can offer valuable insights into the nature of irregularities encountered by vehicles.

4

Data Collection and Implementation

4.1 Data Collection

In order to correctly train the machine learning classifiers, segments of both smooth and corrupted road are necessary. For this purpose, the process of gathering data for this application involved these steps: first, an irregularity is identified on the asphalt in the city of Padova. For each considered irregularity, ten runs of 17.5 seconds were conducted, comprising an initial period of smooth road, followed by the irregularity. The frequency of data acquisition is set at 20Hz, so each measurement comprises of 350 samples. The data is sent via Wi-Fi from the Arduino board attached to the vehicle to the Arduino connected to the computer. The latter receive the data in the form of a *.txt* file with 350 rows and 8 columns. In Table 4.1 is showed an example of two rows of the complete dataset. In the first, second and third column there are respectively the x, y and z component of the acceleration, in the fourth column there are the values of the sonar, in the fifth column there is the speed directly calculated from the latitude and longitude coordinate sent by the GPS module (column six and seven). Lastly, in the eighth column there is a flag that can be either 0 or 1. Since the frequency of acquisition of the GPS signal is lower than the rest of the data, the latitude and longitude data are directly copied from the previous acquisition (flag$= 0$), or a new sample is received from the GPS module (flag$= 1$).

Table 4.1: Example of two rows of the complete dataset.

a_X	a_Y	a_Z	*sonar*	*speed*	*latitude*	*longitude*	*checkBit*
5.32	-8.46	1.65	21.95	1.89	45.408013	11.888756	1
5.73	-7.9	1.37	22.33	1.89	45.408013	11.888756	0

4.1.1 Dataset

In order to get a representative dataset, different types of irregularity have been considered. This subsection displays eight different potholes or bumps taken into consideration (see Fig. 4.1). The

(a) *Photo of irregularity* 1. **(b)** *Photo of irregularity* 2. **(c)** *Photo of irregularity* 3.

(d) *Photo of irregularity* 4. **(e)** *Photo of irregularity* 5. **(f)** *Photo of irregularity* 6.

(g) *Photo of irregularity* 7. **(h)** *Photo of irregularity* 8.

Figure 4.1: Photos of the five irregularity taken into account.

dataset acquired has been formatted and saved in an institutional repository*. The original dataset is organized as follows: there are eight folders, each representing one of the eight different anomalies under consideration, and each folder contains ten *.txt* files, with each file corresponding to one of the ten runs conducted for each anomaly. The windowed dataset used for processing follows a similar structure, with the distinction that within each of the eight folders, there are ten subfolders (one for each run), and each subfolder contains ten *.txt* files, each representing the data for individual windows. A sample of the data acquisition for each irregularity is displayed

*Link where to find the dataset: https://researchdata.cab.unipd.it/id/eprint/1088

in the following figures 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8 and 4.9. In these figures are plotted the three axis acceleration values and the sonar values of one run for each irregularity. As expected, for different anomalies, we observe different profiles in the plot curves.

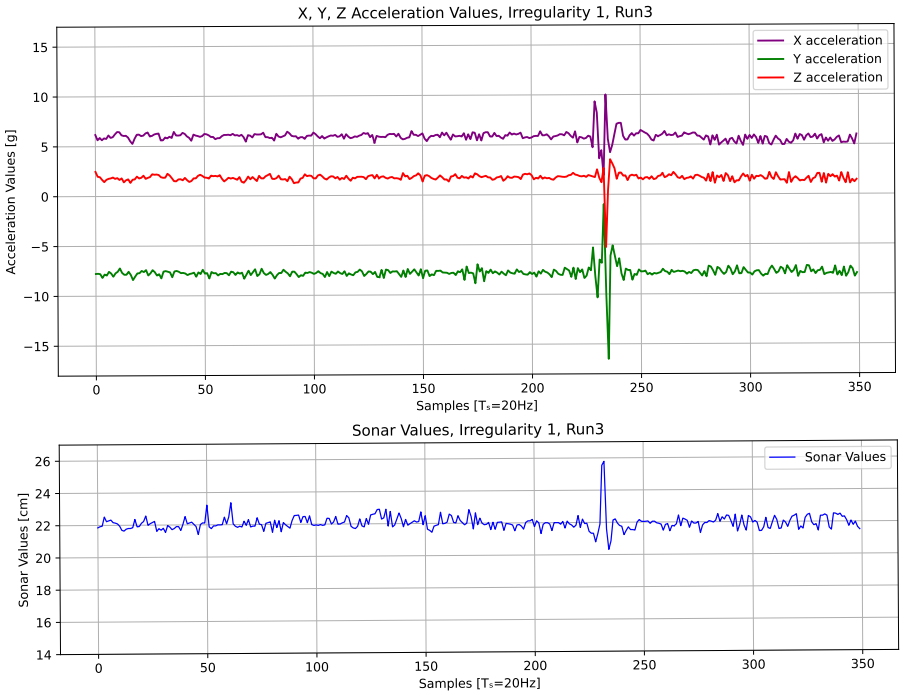


Figure 4.2: Samples of acceleration and sonar values for irregularity one.

In Fig. 4.2 the irregularity has a fairly small duration and inspecting the plot of the sonar values registered, it is apparent that over the manhole there is a sudden increase of the distance between the chassis of the vehicle and the road pavement.
Is interesting also to carefully inspect Fig. 4.4, where the bicycle is traversing a speed bump (Fig. 4.1(c)). From the acceleration trace is evident a sudden shake where the irregularity is located. However, more interesting is again the plot of the sonar values. As one would expect, while crossing a speed bump, the vehicle's center initially experiences an increase in its distance from the ground as the front tire ascends the bump. This is followed by a sudden decrease in the same distance as the tire descends from the bump, ending in a similar process when the rear tire undergoes the same sequence. This succession is clearly evident inspecting the sonar plot in Fig. 4.4.
Fig. 4.5 and Fig. 4.6 illustrate a prolonged duration of the irregularity. Notably, upon closer inspection of the images depicting these irregularities (see Figures 4.1(c) and (d)), it becomes evident that both these two anomalies exhibit a distinct pattern of potholes and cracks rather then a single irregularity as in the previous cases. In conclusion, upon examining the plots in Fig. 4.9, it becomes apparent that when the vehicle traverses gravel, both the accelerometer and sonar traces highlight the uneven terrain.

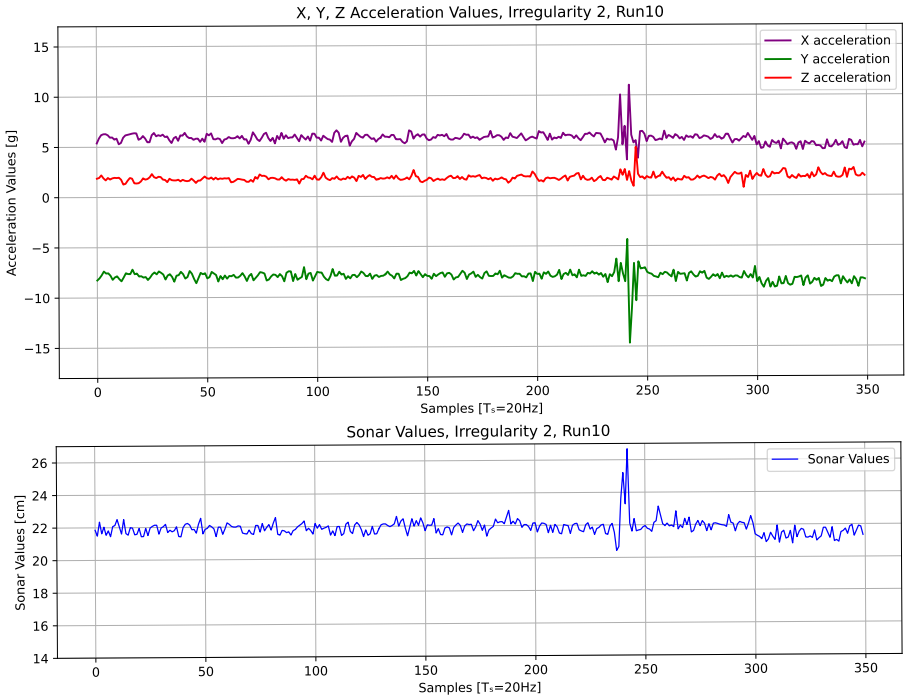


Figure 4.3: Samples of acceleration and sonar values for irregularity two.

4.2 Implementation

The algorithms described in the preceding chapter have been implemented and applied to the collected dataset. The chosen features for classification include the "Peak-to-Peak (P2P)" value and the "standard deviation" of the acceleration trace. The P2P value is straightforwardly determined as the difference between the maximum and minimum values of the accelerations in the window. The standard deviation is defined as $\sigma = \sqrt{\frac{|x-\bar{x}|^2}{N}}$, where x represents individual data points, $\bar{x}$ is the mean, and N is the total number of data points in the window. The rationale behind choosing these features is grounded in the observation that, when a vehicle traverses an irregularity, the acceleration trace exhibits spikes, a characteristic captured by both the P2P value and the standard deviation. The testing process started with using only the P2P value. Subsequently, separate tests were conducted with only the standard deviation. Finally, a comprehensive evaluation was performed by combining both the P2P value and the standard deviation as input measures. This step-by-step testing approach aimed to understand the impact of individual features and their combination on the algorithm's performance in detecting and classifying irregularities in the dataset.

Code snippet 4.1 shows the creation of the first input dataset, containing only the P2P values of the three accelerations.

```
1 p2p = np.zeros((800, 3))
2 max_value = 0
```

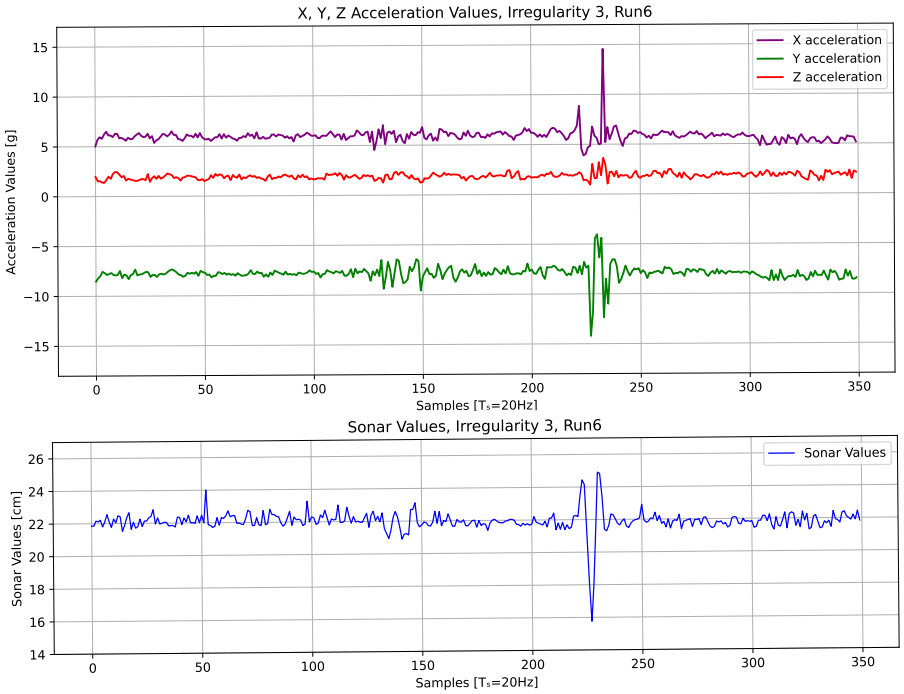


Figure 4.4: Samples of acceleration and sonar values for irregularity three.

```
min_value = 0
p2p_value = 0
Xp2p = []
varAcc2 = ["accX", "accY", "accZ"]
for variableAcc in varAcc2:
 for j in range(1,9):
  for k in range(1, 11):
   for i in range(1, 11):
    max_value = max(windowed[variableAcc][f"dataset{j}"][f"Run{k}"][f"win{i}"])
    min_value = min(windowed[variableAcc][f"dataset{j}"][f"Run{k}"][f"win{i}"])
    p2p_value = max_value - min_value
    Xp2p.append(p2p_value)
for i in range(800):
 p2p[i, 0] = Xp2p[i]
 p2p[i, 1] = Xp2p[i+800]
 p2p[i, 2] = Xp2p[i+1600]
```

Listing 4.1: Python code to create the matrix *p2p* containing the P2P value of the three-axis acceleration for each window.

The final matrix, denoted as $p2p$, comprises the P2P values in its columns. Specifically, the first column represents the P2P values for the x acceleration, the second column denotes the P2P values for the y acceleration, and the third column encapsulates the P2P values for the z acceleration. Similarly, the input matrix containing the standard deviation values for the three acceleration is

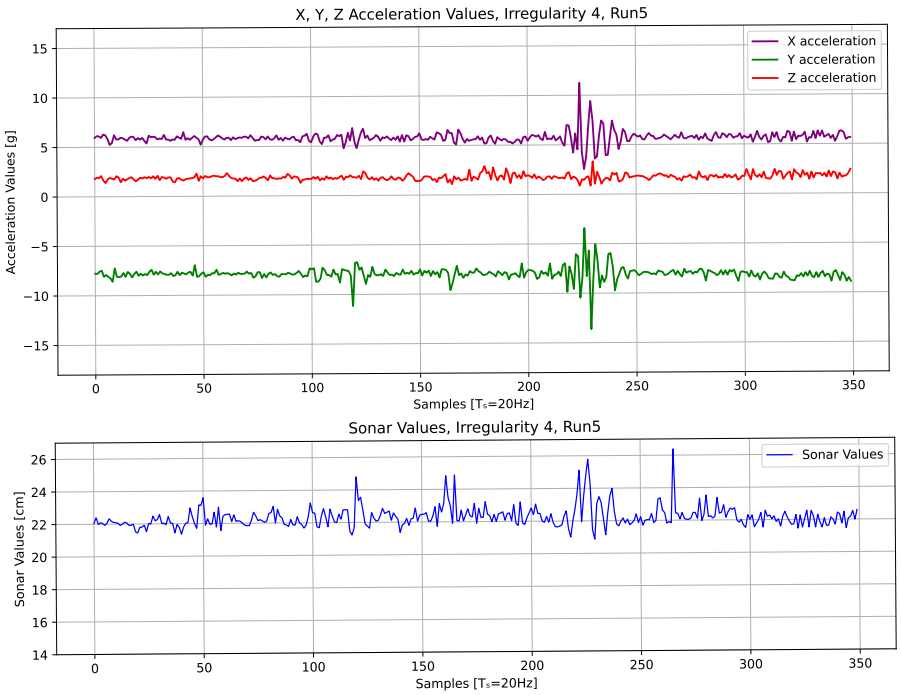


Figure 4.5: Samples of acceleration and sonar values for irregularity four.

constructed. Code snippet 4.2 shows the code to create the *std* input matrix.

```
std = np.zeros((800, 3))
std_value = 0
Xstd = []
varAcc2 = ["accX", "accY", "accZ"]
for variableAcc in varAcc2:
 for j in range(1,9):
  for k in range(1, 11):
   for i in range(1, 11):
    std_value = np.std(windowed[variableAcc][f"dataset{j}"][f"Run{k}"][f"win{i}"
    ])
    Xstd.append(std_value)
for i in range(800):
 std[i, 0] = Xstd[i]
 std[i, 1] = Xstd[i+800]
 std[i, 2] = Xstd[i+1600]
```

Listing 4.2: Python code to create the matrix *std* containing the standard deviation value of the three-axis acceleration for each window.

Finally, following the same principle, the matrix *combined* containing both the P2P and standard deviation values is created (see code snippet 4.3).

```
combined = np.zeros((800, 6))
max_value = 0
```

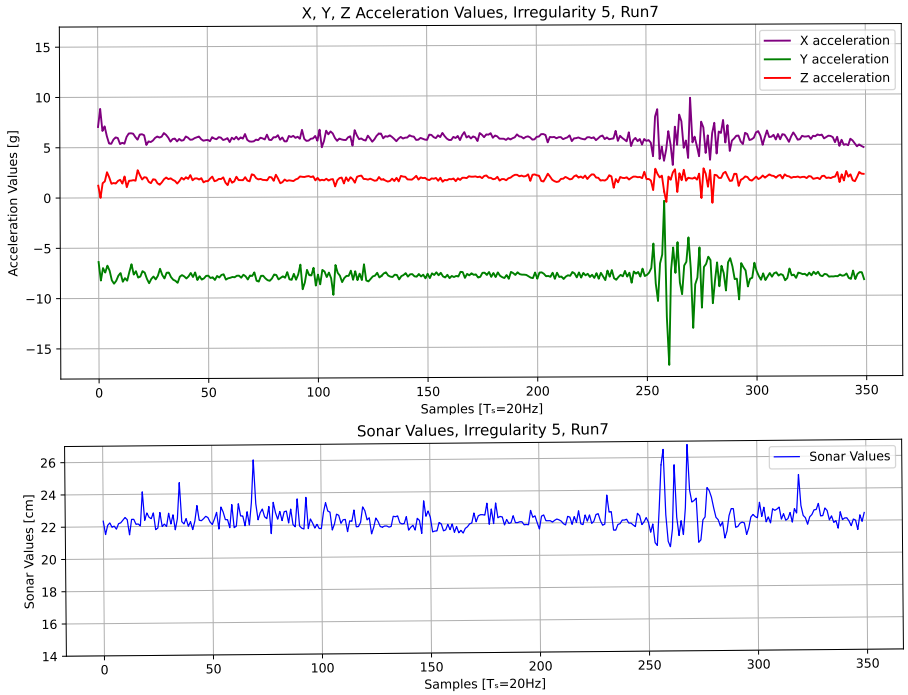


Figure 4.6: Samples of acceleration and sonar values for irregularity five.

```
min_value = 0
p2p_value = 0
std_value = 0
XCombined = []
varAcc2 = ["accX", "accY", "accZ"]
for variableAcc in varAcc2:
 for j in range(1,9):
  for k in range(1, 11):
   for i in range(1, 11):
    max_value = max(windowed[variableAcc][f"dataset{j}"][f"Run{k}"][f"win{i}"])
    min_value = min(windowed[variableAcc][f"dataset{j}"][f"Run{k}"][f"win{i}"])
    p2p_value = max_value - min_value
    XCombined.append(p2p_value)
for variableAcc in varAcc2:
 for j in range(1,9):
  for k in range(1, 11):
   for i in range(1, 11):
    std_value = np.std(windowed[variableAcc][f"dataset{j}"][f"Run{k}"][f"win{i}"
    ])
    XCombined.append(std_value)
for i in range(800):
 combined[i, 0] = XCombined[i]
 combined[i, 1] = XCombined[i+800]
 combined[i, 2] = XCombined[i+1600]
```

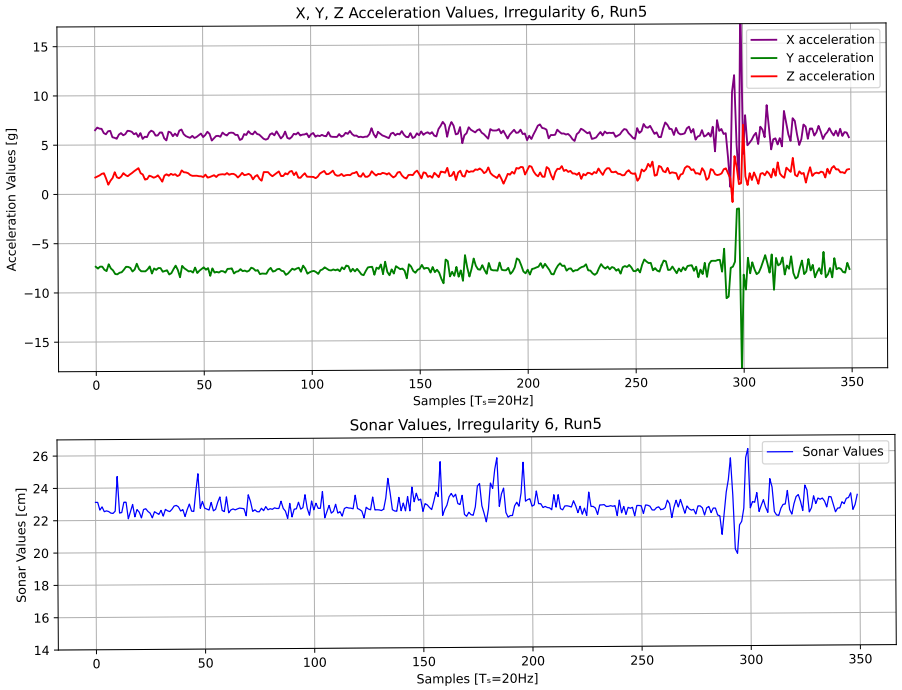


Figure 4.7: Samples of acceleration and sonar values for irregularity six.

```
combined[i, 3] = XCombined[i+2400]
combined[i, 4] = XCombined[i+3200]
combined[i, 5] = XCombined[i+4000]
```

Listing 4.3: Python code to create the matrix *combined* containing both the P2P and standard deviation values.

The *training* and *test* sets have been generated using the *train_test_split*() function from the scikit-learn library. This function takes in input the parameters listed in Table 4.2. For all three different kinds of input matrices, the *test set* is set to be 30% of the whole dataset, comprising 240 windows in total. The *random_state* variable is set to 42 so to have randomized training and test set in input to the algorithms.

An essential factor influencing the performance of these models is their hyperparameters. Optimal values for these hyperparameters can significantly enhance a model's performance. GridSearchCV is a process employed for hyperparameter tuning, aimed at determining the most effective values for a given model. It is important to note that anticipating the best values for hyperparameters in advance is challenging. Ideally, exploring all possible values is necessary to identify the optimal configuration. However, manually conducting such an exhaustive search can be time-consuming and resource-intensive. Therefore, *GridSearchCV*() is utilized to automate the hyperparameter tuning process. *GridSearchCV*() is a function within scikit-learn's *model_selection* package. This function systematically iterates through predefined hyperparameters, fitting the estimator on the training set for each combination. In the end, it allows us to select the best pa-

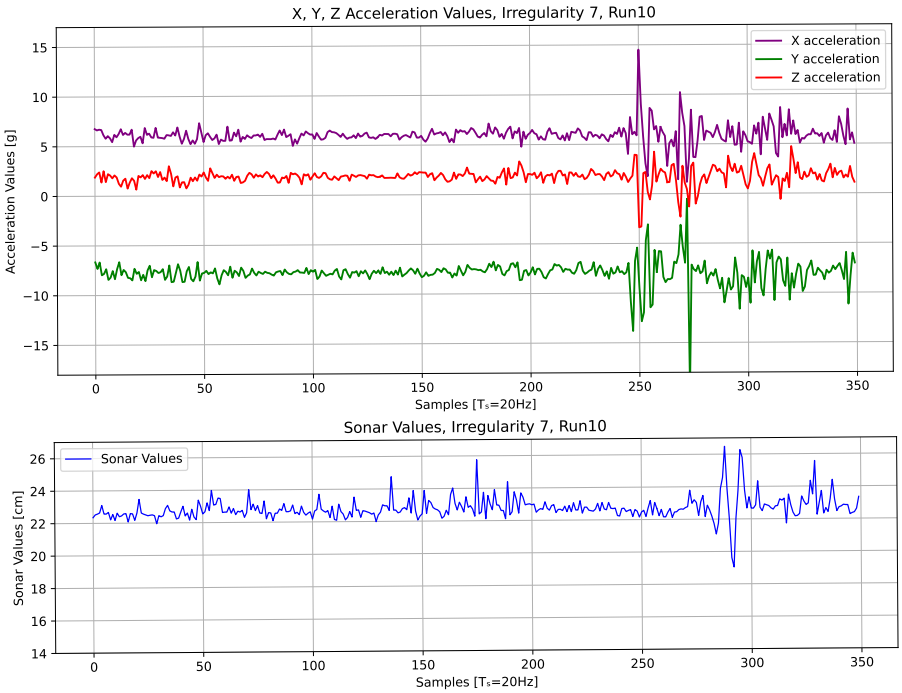


Figure 4.8: Samples of acceleration and sonar values for irregularity seven.

rameters from the specified hyperparameter options. This automated approach streamlines the hyperparameter tuning process and helps in identifying the most effective configuration for the model. This function takes in input the parameters listed in the Table 4.3.

As mentioned earlier, SVM and RF are supervised machine learning techniques, meaning that in order to make predictions, they must be first trained using labeled data. The analysis is divided into two parts. In the first part the models are employed for binary classification, aiming to predict whether the window considered represent smooth road or an irregularity. The analysis proceed then adding another layer of complexity trying to discriminate also the severity of the irregularity. For this reason, two matrices containing the correct labels have been created. The first matrix, denoted as Y contains the label for binary classification. Is a matrix filled with 0 or 1, respectively representing smooth road or irregularity. The matrix $Y3$, in contrast, contains the labels for multiclass classification, namely 0 for smooth road, 1 for low severity irregularity and 2 for high severity irregularity. The code to create these matrices is displayed in code snippet 4.4.

```
Y = np.full(800,99)
n=0
for j in range(9):
 for k in range(1, 11):
  for i in range(1, 11):
   Y[n] = int(windowed["YN"][f"dataset{j}"][f"Run{k}"][f"win{i}"][0])
   n = n+1
Y3 = np.full(800,99)
```

Table 4.2: Parameters in input to the *train_test_split()* function.

Parameter	Explanation
array: array-like	Input array
test_size: float, default = None	between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split
random_state: int, default = None	Controls the shuffling applied to the data before applying the split
shuffle: bool, default=True	Whether or not to shuffle the data before splitting
stratify: array-like, default = None	If not None, data is split in a stratified fashion, using this as the class labels

Table 4.3: Parameters in input to the *train_test_split()* function.

Parameter	Explanation
estimator: estimator object	model instance for which you want to check the hyperparameters
param_grid: dict or list of dictionaries	Dictionary with parameters names (str) as keys and lists of parameter settings to try as values
scoring: str	Strategy to evaluate the performance of the cross-validated model on the test set
n_jobs: int, default=None	Number of jobs to run in parallel
refit: bool, str, or callable, default=True	Refit an estimator using the best found parameters on the whole dataset
cv: int, cross-validation generator or an iterable, default=None	Determines the cross-validation splitting strategy
verbose: int	Controls the verbosity: the higher, the more messages

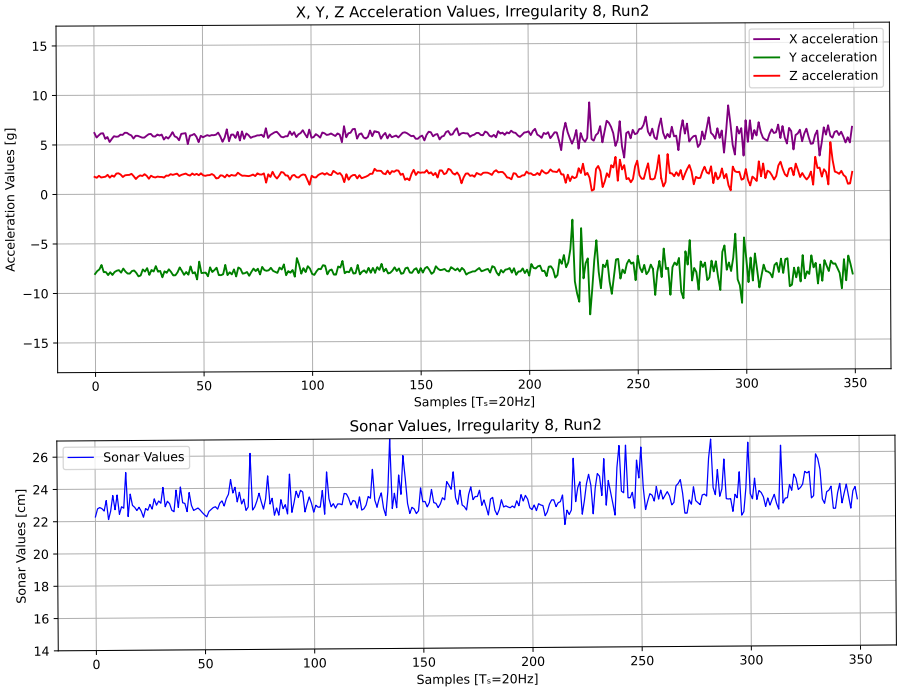


Figure 4.9: Samples of acceleration and sonar values for irregularity eight.

```
n=0
for j in range(9):
 for k in range(1, 11):
  for i in range(1, 11):
   Y3[n] = int(windowed["severity"][f"dataset{j}"][f"Run{k}"][f"win{i}"][0])
   n = n+1
```

Listing 4.4: Cretion of the matrix *Y* and *Y3* containing the labels for binary and multiclass classification respectively.

5 SVM Classifier – Results

This chapter details the results when the dataset is inputted to the SVM classifier. The SVM classifier function *SVC*() from the scikit-learn in python takes in input the parameters listed in the Table 5.1 and the algorithm has been trained and tested with different input configurations as detailed in the previous chapter. This chapter is divided into two main sections, one concerning the binary classification and one the multiclass classification. These section are then further divided into three subsections each, separating the analysis carried out for the three different input matrices considered.

5.1 Binary Classification

5.1.1 Only *p2p* Matrix

In this section are displayed the results of the SVM binary classifier when the input is the *p2p* matrix. Recall that the feature considered in this case is the *P2P* value for all three acceleration for each window.

Linear Kernel

First, the algorithm has been tested with a linear kernel. The code to define and train the classifier is displayed in code snippet 5.1.

```
# parameters for linear SVM
parameters = {'C': [ 0.01, 0.1, 1, 10], 'gamma':[0.01,0.1,1.0]}

#train linear SVM
svc_p2p = SVC(kernel = 'linear')
clf_p2p = GridSearchCV(svc_p2p, parameters, cv = 4, n_jobs=-1)
```

Table 5.1: Parameters in input to the *train_test_split()* function.

Parameter	Explanation
C: float, default=1.0	Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty
kernel: "linear", "poly", "rbf", "sigmoid", "precomputed" or callable, default="rbf"	Specifies the kernel type to be used in the algorithm
degree: int, default=3	Degree of the polynomial kernel function ("poly"). Must be non-negative. Ignored by all other kernels
gamma: "scale", "auto" or float, default="scale"	Kernel coefficient for "rbf", "poly" and "sigmoid"
coef0: float, default=0.0	Independent term in kernel function. It is only significant in "poly" and "sigmoid"
shrinking: bool, default=True	Whether to use the shrinking heuristic
probability: bool, default=False	Whether to enable probability estimates
tol: float, default=$1e-3$	Tolerance for stopping criterion
*cache_size: float, default=*200	Specify the size of the kernel cache (in MB)
class_weight: dict or "balanced", default=None	Set the parameter *C* of class *i* to *class_weight*$[i] * C$ for SVC
verbose: bool, default=False	Enable verbose output
probability: bool, default=False	Whether to enable probability estimates
max_iter: int, default=-1	Hard limit on iterations within solver, or -1 for no limit
decision_function_shape: "ovo", "ovr", default="ovr"	Whether to return a one-vs-rest ("ovr") decision function of shape (*n_samples*, *n_classes*) as all other classifiers, or the original one-vs-one ("ovo") decision function of libsvm which has shape ($n\_samples, n\_classes * (n\_classes - 1)/2$)
break_ties: bool, default=False	If true, *decision_function_shape* $= ovr$, and number of classes > 2, predict will break ties according to the confidence values of *decision_function*; otherwise the first class among the tied classes is returned
random_state: int, RandomState instance or None, default=None	Controls the pseudo random number generation for shuffling the data for probability estimates

```
clf_p2p.fit(X_train_p2p,y_train_p2p)
```

Listing 5.1: Definition and training of the SVM classifier with a linear kernel. The input is the *p*2*p* matrix (see previous chapter).

The code shows the definition and training of the SVM algorithm when inputted with the *p*2*p* matrix detailed in the previous chapter. The *GridSearchCV*() function takes in input in this case the *C* and *gamma* parameters. The *C* parameter is the inverse of the λ parameter explained in Chapter 3 and trades off correct classification of training examples against maximization of the decision function's margin. For larger values of *C*, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower *C* will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy. In other words *C* behaves as a regularization parameter in the SVM. The *gamma* parameter defines how far the influence of a single training example reaches, with low values meaning "far" and high values meaning "close". It can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. The cross-validation is set to 4, meaning that the training set is divided into four sets. Three of them are used for training and the remaining one for the validation. This process is executed four times, each time selecting a different set used for validation. The best score resulting from the search is 0.9679 with the parameter *C* set to 10 and *gamma*= 0.01.

Polynomial Kernel - Order 2

The same reasoning is applied , with the only differences being the kernel of the classifier, now set to "poly". The best results are found with the *C* parameter set to 0.01 and the *gamma* value equal to 1.0. The accuracy in this case is 0.9661.

Polynomial Kernel - Order 3

Similarly to the previous subsection, the *GridSearchCV*() is applied to the model with the polynomial kernel of order three. The best results are found with the value of *C*= 0.01 and *gamma*= 0.1. The result is an accuracy of 0.9678.

Radial Basis Function Kernel

In the same way, the model has been also trained with the *"rbs"* kernel, reaching a training accuracy of 0.9660 with *gamma*= 0.01 and *C*= 1.

The best results for this type of input is found with the polynomial kernel of order 3. In this case, the algorithm achieved a training error of 0.034 and a test error of 0.0041. The model properly classified all the smooth road instances and 45 out of 46 samples labeled as 1.

5.1.2 Only *std* Matrix

The same process has been carried out with the *std* matrix as the input to the classifier. Recall that the matrix contains the value of the standard deviation of the three accelerometer traces for

each window. The results and the optimal parameters found with the *GridSearchCV*() function are displays in the Table 5.2. The model with the *rbs* kernel is selected to be the best model in

Table 5.2: Results for the models when the input is the *std* matrix.

Kernel	C	*gamma*	accuracy
Linear	10	N/A	0.9768
Polynomial 2	1	1.0	0.9786
Polynomial 3	1	1.0	0.9767
Radial Basis Function	10	1.0	0.9803

this case. It achieved a training and test error respectively of 0.0196 and 0.0083 with only 2 false positive and 0 false negative instances.

5.1.3 *p2p* and *std* Matrix Combined

In the end, the same models have been trained and tested when the input is the matrix *combined*. Once again, the results are shown in the Table 5.3 below. With the *combined* matrix as input to

Table 5.3: Results for the models when the input is the *combined* matrix.

Kernel	C	*gamma*	accuracy
Linear	10	N/A	0.9875
Polynomial 2	0.1	1.0	0.9804
Polynomial 3	1	0.1	0.9766
Radial Basis Function	100	0.01	0.9767

the classifier, the best model is the one with the *linear* kernel. With this configuration the training error reads 0.0107 and the test error is equal to 0.0042. Out of all the samples in input to the algorithm, only 1 instance labeled as 0 is wrongly classified as 1.

5.2 Multiclass Classification

Having demonstrated the algorithm's proficiency in discriminating between smooth roads and irregularities, subsequent experiment was conducted to introduce additional complexity. The algorithm underwent training with three distinct labels, 0, 1, and 2, where 0 designates a smooth road, 1 characterizes a smooth irregularity, and 2 signifies a substantial irregularity. The code employed for training and testing is the same to the procedures used in binary classification, with the only difference being the number of different labels inside the $Y3$ array. The results for the different models are shown in the next subsections.

Table 5.4: Results for the models when the input is the *p2p* matrix.

Kernel	C	*gamma*	accuracy
Linear	0.1	N/A	0.9125
Polynomial 2	0.1	.01	0.9107
Polynomial 3	.0.01	1	0.9161
Radial Basis Function	10	0.1	0.9196

5.2.1 Only *p2p* Matrix

The analysis started once again considering the *p2p* matrix as input to the algorithm. The SVM classifier has been tested with different kernels and different parameters thanks to the *GridSearch*() function. The results for the different kernel functions are displayed in the Table 5.4. The algorithm with the *rbs* kernel is chosen as the optimal configuration in this case, although the other configurations also yield fairly high results. The model exhibits a training error equal to 0.041071 and a test error equal to 0.091667. Specifically, all samples associated with label 0, representing smooth road segments, are correctly classified. However, the algorithm exhibits less accuracy in correctly classifying irregularity segments. Out of the 30 samples associated with label 1, 2 are misclassified as 0, and 7 are misclassified as 2. Additionally, out of the 16 instances of label 2, only 3 are correctly labeled as 2, while the remaining 13 are erroneously labeled as 1. The confusion matrix in Fig. 5.1 summarize the results.

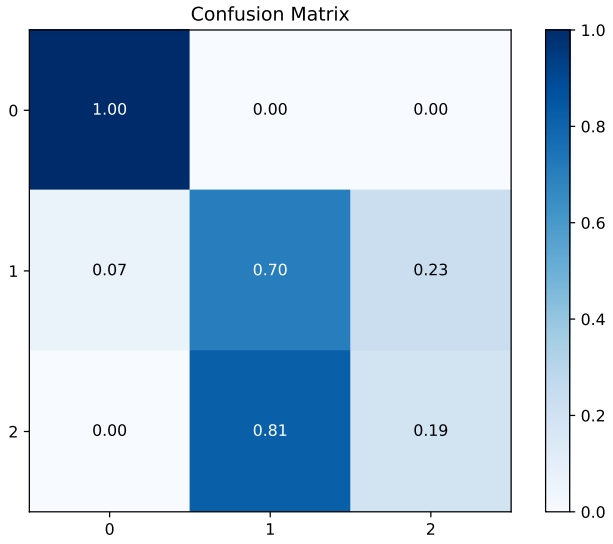


Figure 5.1: Confusion matrix when the input is the *p2p*. Multiclass classification.

Only *std* Matrix

The analysis continued considering the *std* matrix as input to the algorithm, and, with the same reasoning, the results with the different parameters are summarized in the Table 5.5. The best

Table 5.5: Results for the models when the input is the *p2p* matrix.

Kernel	C	*gamma*	accuracy
Linear	10	N/A	0.9303
Polynomial 2	1	1.0	0.9321
Polynomial 3	1	1	0.9285
Radial Basis Function	10	0.1	0.9303

configuration in this case is the one with the polynomial kernel of order 2. The training error is 0.066 and the test error is 0.071. Still the algorithm struggles to classify the two types of irregularity, achieving an accuracy of 19% in recognizing high severity irregularity. In the Fig. 5.2 is displayed the confusion matrix for this model.

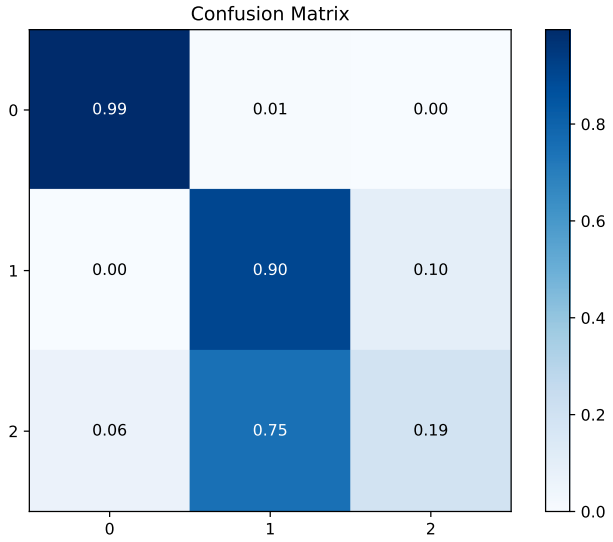


Figure 5.2: Confusion matrix when the input is the *std*. Multiclass classification.

p2p and *std* Matrices Combined

The analysis concluded combining the two matrices and inputting in the algorithm the matrix *combined*. The results are shown in the Table 5.6. Once again, the best results is found with the polynomial of order 2 kernel function. In this case, 1 out of the 194 instances of smooth road is wrongly classified as low severity irregularity, 9 out of 30 samples are wrongly assigned the label

Table 5.6: Results for the models when the input is the *combined* matrix.

Kernel	C	*gamma*	accuracy
Linear	10	N/A	0.9339
Polynomial 2	1	1.0	0.9357
Polynomial 3	0.01	1	0.9339
Radial Basis Function	10	0.1	0.9250

2 and 5 out of 16 high severity irregularity are misclassified. The Fig. 5.3 shows visually these results.

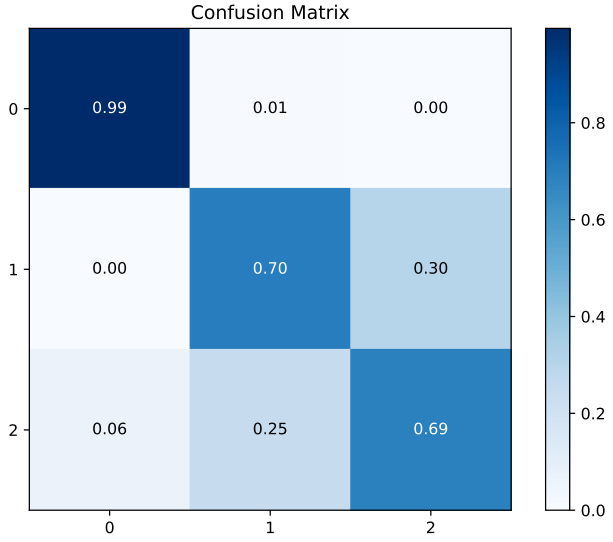


Figure 5.3: Confusion matrix when the input is the *combined*. Multiclass classification.

6
Random Forest – Results

In this chapter, the results of the RF classifier are listed and explained. The RF classifier in Python takes in input some parameter explained in Table 6.1. Similar to the SVM classifier, the RF algorithm underwent training and testing with the three configurations of inputs described earlier. The analysis started with binary classification (distinguishing between smooth road and irregularity) and then proceeded to explore the analysis of irregularity severity. The results for all these analyses are presented in the subsequent subsections.

6.1 Binary Classification

6.1.1 Only *p2p* Matrix

In alignment with the prior analysis, the function *GridSearchCV*() facilitated the identification of optimal parameters that yielded greater accuracy in the model's predictions. The parameters considered in this case are the *"max_depth"* of the tree, the *"min_sample_leaf"* parameter, and the *"n_estimators"* parameter. The code to run the *GridSearch*() function is displayed in the cose snippet 6.1.

```
RF_p2p_GS = RandomForestClassifier(random_state=42, n_jobs=-1)
params = {'max_depth': [5,10,20],
          'min_samples_leaf': [2,5,10,20,50,100,200],
          'n_estimators': [10,25,30,50,100,200]
         }
grid_search_p2p = GridSearchCV(estimator=RF_p2p_GS,
                               param_grid=params,
                               cv = 4,
                               verbose=1, scoring="accuracy")
grid_search_p2p.fit(X_train_p2p, y_train_p2p)
```

Table 6.1: Parameters in input to the *RandomForestClassifier()* function.

Parameter	Explanation
n_estimators: int, default = 100	Number of trees in the forest
criterion: default = "gini"	Function that measure the quality of the split
max_depth: int, default = None	Maximum depth of the tree
min_samples_split: int or float, default=2	Minimum number of samples required to split an internal node
min_samples_leaf: int or float, default = 1	Minimum number of samples required to be at a leaf node
min_weight_fraction_leaf: float, default = 0.0	Minimum weighted fraction of the sum of weights required to be at a leaf node
max_features: int or float, default = "sqrt"	Number of features to consider when looking for the best split (if *"sqrt"*, then $max\_features=\sqrt{n\_features}$
max_leaf_nodes: int, default = None	Grow trees with *max_leaf_nodes* in best-fashion
min_impurity_decrease: float, default = 0.0	A node will be split if the split induces a decrease of the impurity greater or equal to this value
bootstrap: bool, default = True	Whether boostrap samples are used when building the trees
obb_score: bool, default = False	Whether to use the out-of-bag samples to estimate the generalization score
n_jobs: int, default = None	Number of jobs to run in parallel
random_state: int, default = None	Controls the randomness of the bootstrapping of the samples used when building the trees
verbose: int, default = 0	Controls the verbosity when fitting and predicting
warn_start: bool, default = False	When set to *True*, reuse the solution of the previous call to fit and add more estimators to the ensemble
class_weight: dict or list of dicts, default = None	Weight associated with classes
ccp_alpha: non-negative float, default = 0.0	Complexity parameter used for Minimal Cost-Complexity Pruning
max_samples: int or float, default = None	If bootstrap is *True*, the number of samples to draw from X to train each base estimator

```
11 RF_best_p2p = grid_search_p2p.best_estimator_
```

Listing 6.1: Definition and training of the RF classifier. The input is the p2p matrix.

In the final line of code, the optimal classifier is selected and stored in the *RF_best_p2p* variable. In this case, the best performance is achieved with *max_depth*= 5, *min_samples_leaf*= 5, and *n_estimators*= 100. With this configuration, the algorithm demonstrates a training error of 0.0268 and a test error of 0.0. Consistent with the results observed for the SVM classifier, the algorithm performs exceptionally well for binary classification. Figure 6.1 showcases one of the 100 decision trees constituting the forest.

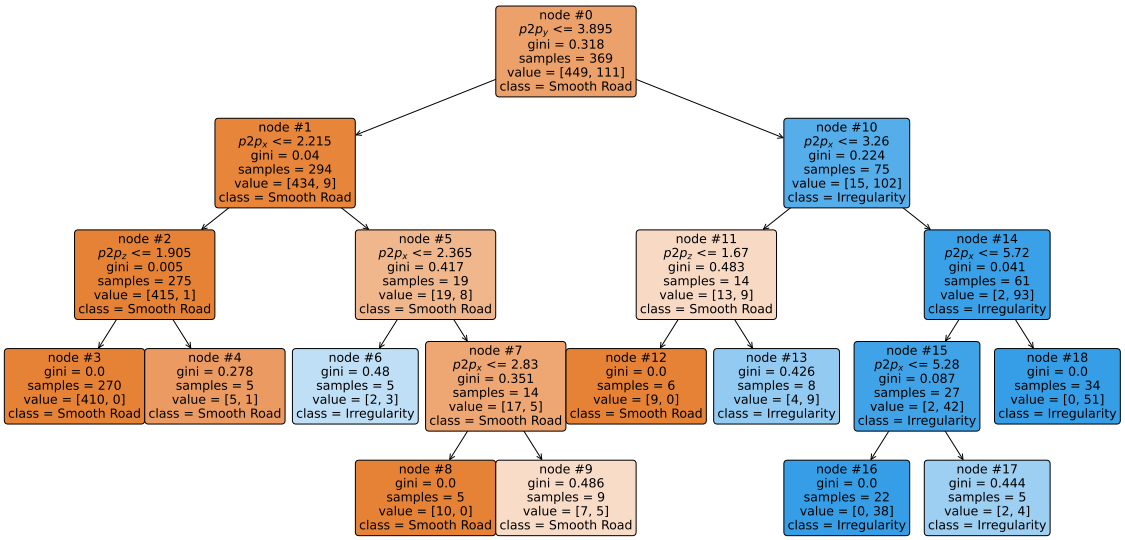


Figure 6.1: One of the 100 decision tree in constituting the random forest. The input is the *p2p* matrix.

6.1.2 Only *std* Matrix

The classifier now takes the *std* matrix as input. The *GridSearch*() function identified the best parameters in this case as *max_depth*= 2, *min_samples_leaf*= 10, and *n_estimators*= 100. The model achieved a training error of 0.0196 and a test error of 0.0124. Figure 6.2 illustrates one of the trees constituting the forest.

6.1.3 *p2p* and *std* Matrices Combined

The analysis concluded by once again considering the combination of the two feature matrices *p2p* and *std*. The parameters yielding the best result are *max_depth*= 5, *min_samples_leaf*= 10, and *n_estimators*= 50. The algorithm exhibits a training error of 0.01785 and a test error of 0.01249. Consistently with the previous configurations, Fig. 6.3 depicts one of the 50 decision trees constituting the random forest.

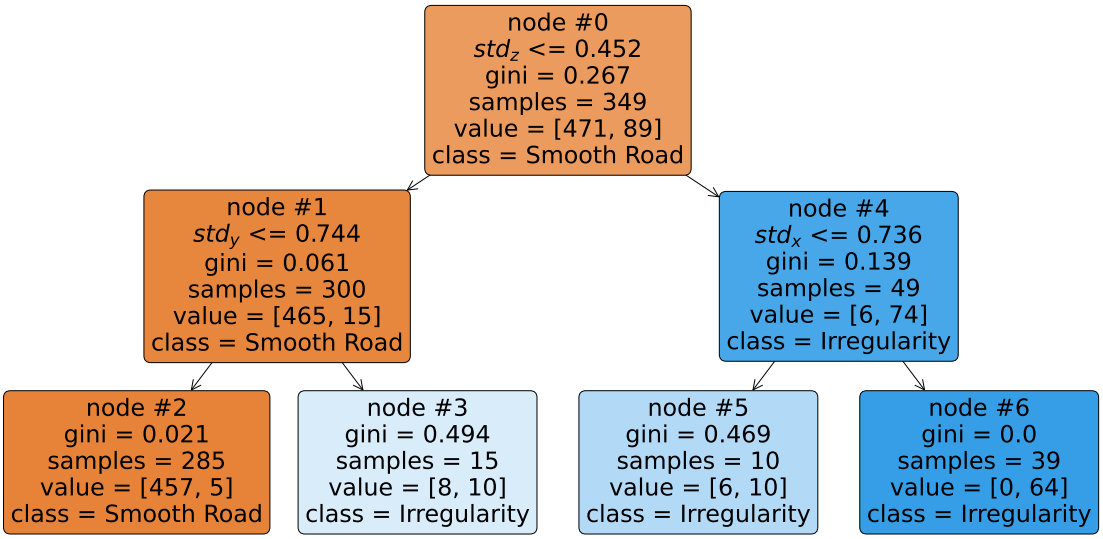


Figure 6.2: One of the 100 decision tree in constituting the random forest. The input is the *std* matrix.

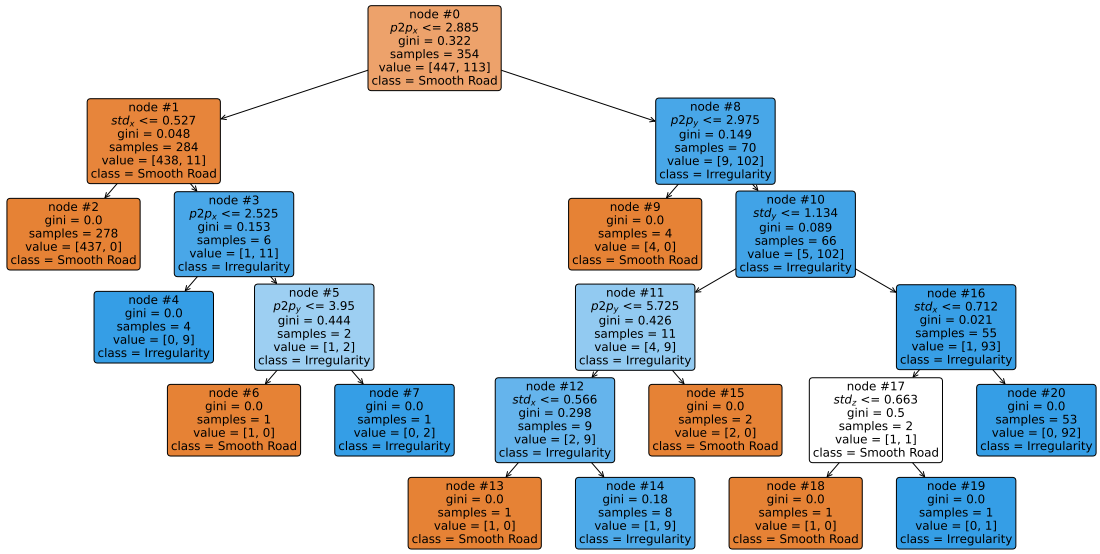


Figure 6.3: One of the 50 decision tree in constituting the random forest. The input is the *combined* matrix.

6.2 Multicalss Classification

Similarly to the analysis carried out for the SVM classifier, the RF algorithm has also been tested for multiclass classification, aiming at classify different irregularity based on their severity.

6.2.1 Only *p2p* Matrix

In this first part the input is the same *p2p* matrix as the previous analysis. The best parameters are *max_depth*= 5, *min_samples_leaf* = 5 and *n_estimators*= 100. The model achieved an accuracy of 0.933, the training error reads 0.055 and the test error is 0.67. The confusion matrix in Fig. 6.4 helps to visualize the results. In particular, the algorithm correctly classify almost all instances corresponding to smooth road, good results are shown for the classification of low severity irregularity, while, as expected, the algorithm with this input and parameters configurations struggles to classify high severity irregularity segments. An example of one of the 100 trees constituting the

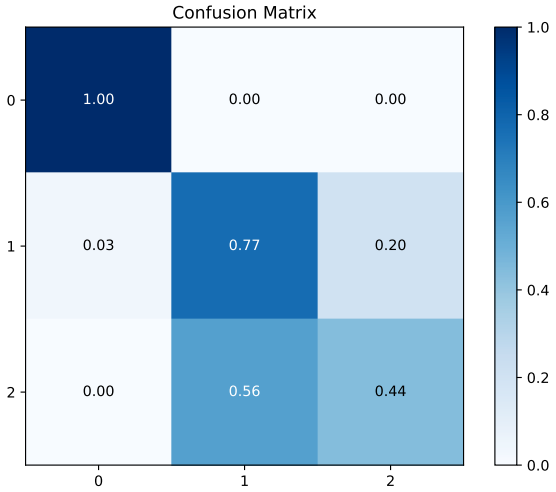


Figure 6.4: Confusion matrix when the input is the *p2p*. Multiclass classification.

forest is displayed in Fig. 6.5.

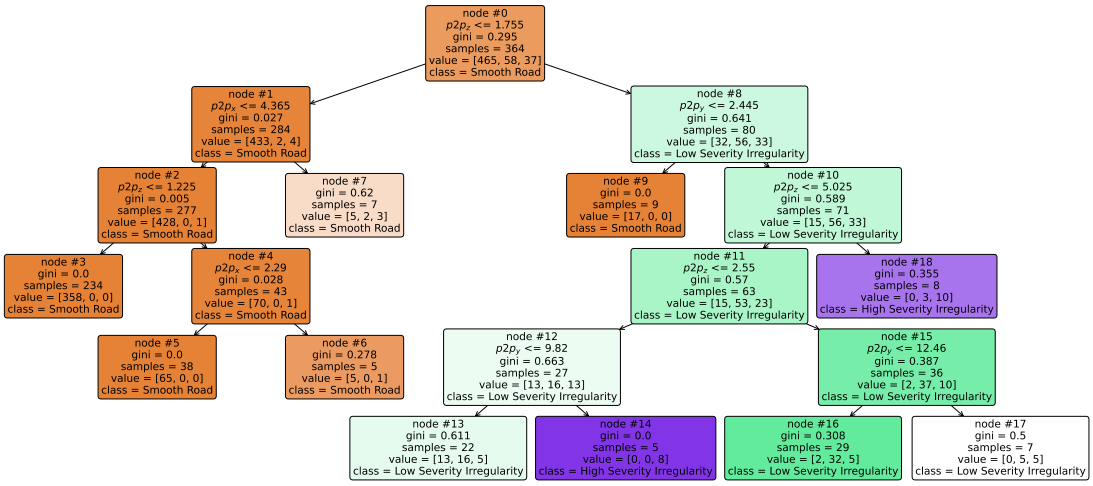


Figure 6.5: One of the 100 decision tree in constituting the random forest. The input is the *p2p* matrix.

6.2.2 Only *std* Matrix

The analysis continued considering the *std* matrix as input to the classifier. The best parameters in output to the *GridSearch*() function are *max_depth*= 5, *min_samples_leaf*= 5 and *n_estimators*= 25. The algorithm exhibited a training accuracy of 0.952 (training error equal to 0.048) and a test accuracy of 0.917 (with a test error of 0.083). Similarly to the previous results, the algorithm is less accurate in the recognition of high severity irregularity, as displayed in the confusion matrix in Fig. 6.6. A decision tree in the forest considered to make the prediction on the test set is displayed

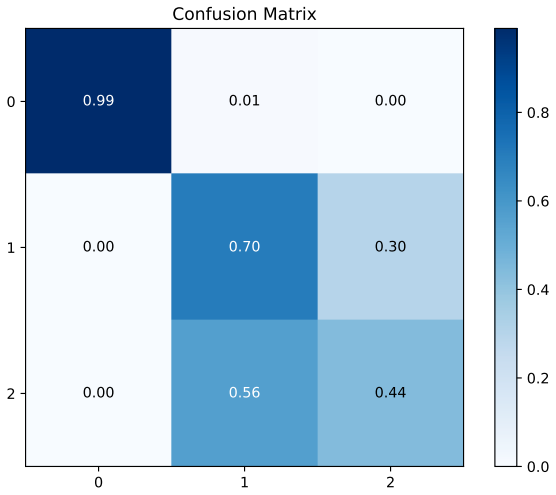


Figure 6.6: Confusion matrix when the input is the *std*. Multiclass classification.

in Fig. 6.7

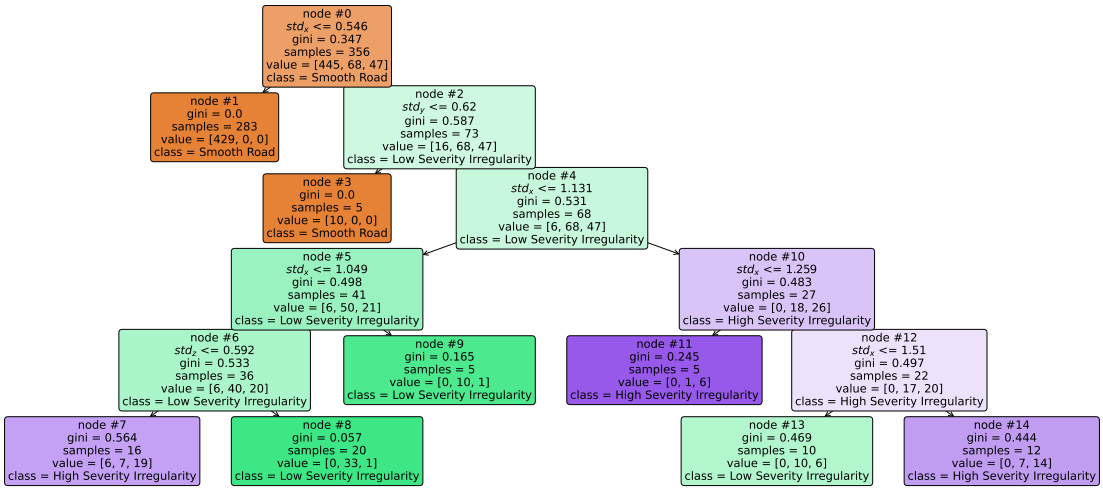


Figure 6.7: One of the 25 decision tree in constituting the random forest. The input is the *std* matrix.

6.2.3 *p2p* and *std* Matrices Combined

The analysis once again concluded considering the *combined* matrix containing both the *P2P* and standard deviation measures for all the windows. In this case the algorithm showed a training error of 0.0357 and a test error of 0.0708, with of over 99% correct classification for instances labeled as 0, 77% for samples labeled as 1 and 50% windows labeled as 2, as depicted in Fig. 6.8. The parameters selected by the *GridSearch*() function are *max_depth*= 5, *min_samples_leaf* = 5 and *n_estimators*= 30. As usual, one of the 30 decision trees that form the random forest is displayed

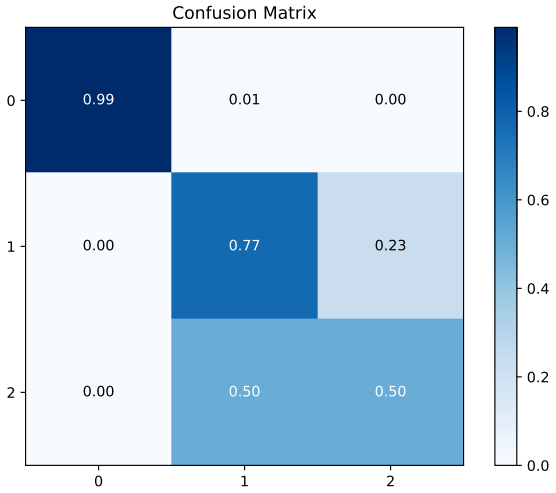


Figure 6.8: Confusion matrix when the input is the *combined*. Multiclass classification.

in Fig. 6.9

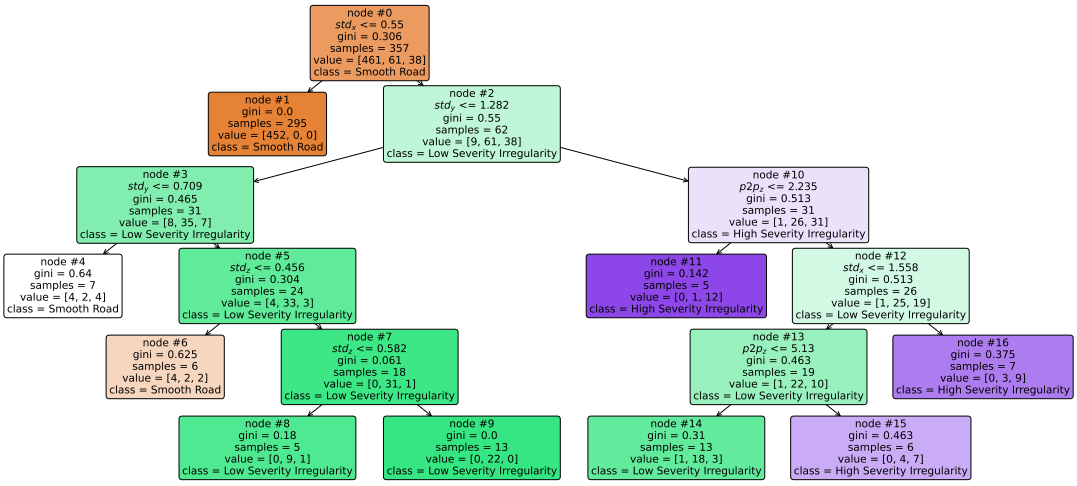


Figure 6.9: One of the 30 decision tree in constituting the random forest. The input is the *combined* matrix.

7
K-Means – Results

The last algorithm trained and tested with the gathered dataset is the *K-Means* clustering technique. The details of the implementation and the results achieved are listed in this chapter. The *KMeans*() function implemented in the scikit-learn library takes in input the parameters listed in the Table 7.1. The K-Means algorithm is sensitive to feature scaling. Therefore, before inputting the matrix into the algorithm, the vectors were standardized. This involved removing the mean and scaling them to have unit variance, using the *StandardScaler*() function from the Python scikit-learn library. This function simply computes $\boldsymbol{z} = (\boldsymbol{x} - \bar{\boldsymbol{x}})/\sigma$, where $\bar{\boldsymbol{x}}$ is the mean and σ is the standard deviation of the vectors, to have the standardized score of each sample of the dataset. The standardized matrix will be the input of the K-Means algorithm.

7.1 Binary Classification

The algorithm was initially tested for binary classification. As an unsupervised machine learning technique, specifying binary classification involves setting the number of clusters to $k = 2$ (recall that the *KMeans*() function necessitates prior knowledge of the number of clusters to partition the points). The code to train the classifier for binary classification is displayed in code snippet 7.1.

```
# Defying KMeans
kmeans = KMeans(n_clusters = 2, init='random', n_init=100, max_iter=500)
kmeans.fit(p2p)
# Store the assigned labels
cluster_assignments = kmeans.labels_
```

Listing 7.1: Definition of the K-Means classifier function.

In the variable "*cluster_assignments*", the labels representing the final cluster assignments of each point are stored. Fig. 7.1 depicts the ground truth corresponding to the 800 analyzed windows,

Table 7.1: Parameters in input to the *RandomForestClassifier()* function.

Parameter	Explanation
n_clusters: int, default=8	Number of clusters to form as well as the number of centroids to generate
init: "k-means++", "k-random", callable or array-like of shape (*n_clusters, n_features*), *default="k-means++"*	Method for initialization
n_init: "auto" or int, default=10	Number of times the k-means algorithm is run with different centroid seeds
max_iter: int, default=300	Maximum number of iterations of the k-means algorithm for a single run
tol: float, default=$1e-4$	Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence
verbose: int, default=0	Verbosity mode
random_state: int, RandomState instance or None, default=None	Determines random number generation for centroid initialization
copy_x: bool, default=True	When pre-computing distances it is more numerically accurate to center the data first
algorithm: "lloyd", "elkan", "auto", "full", default="lloyd"	K-means algorithm to use

where the binary labels are assigned as follows: 0 for smooth road and 1 for irregularity. Instances of smooth roads are represented by red dots, primarily concentrated in the lower region of the graph, reflecting lower values across the three acceleration features. In contrast, irregularities are characterized by elevated P2P values, situating them in the upper part of the plot.

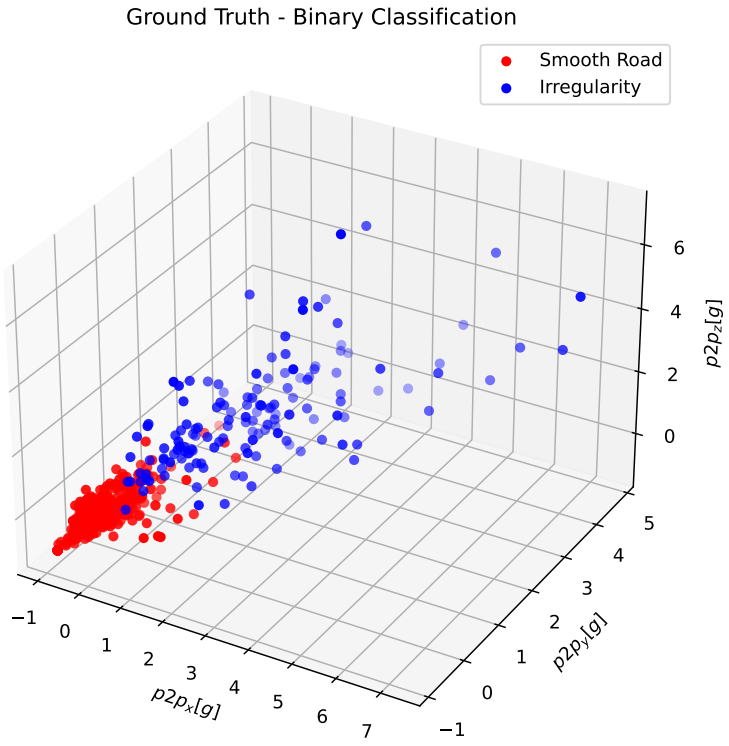


Figure 7.1: Ground truth with the *p2p* matrix as input and $k = 2$.

7.1.1 Only *p2p* Matrix

The analysis started considering the *p2p* matrix as input to the classifier. The *init* parameter is set to *"random"*, meaning that the initial controid seeds are chosen at random between all the possible allocations. The algorithm is then executed $n\_init = 10$ times with different random centroid seeds. The result is displayed in the Fig. 7.2. The instances labeled as 0, corresponding to segments of smooth road, are represented by the red dots on the plot, while the blue dots depict segments with road irregularities. The green stars indicate the two cluster centers identified upon algorithm convergence. As anticipated, instances with low P2P feature values tend to form clusters, whereas those with higher values constitute a distinct group, aligning with expectations. Notably, the model effectively clusters over 99% of the windows associated with smooth road conditions and achieves an accuracy of 85% in classifying windows representing irregularities. The analysis reveals a total of 6 instances classified as "false positives" (cases where the algorithm

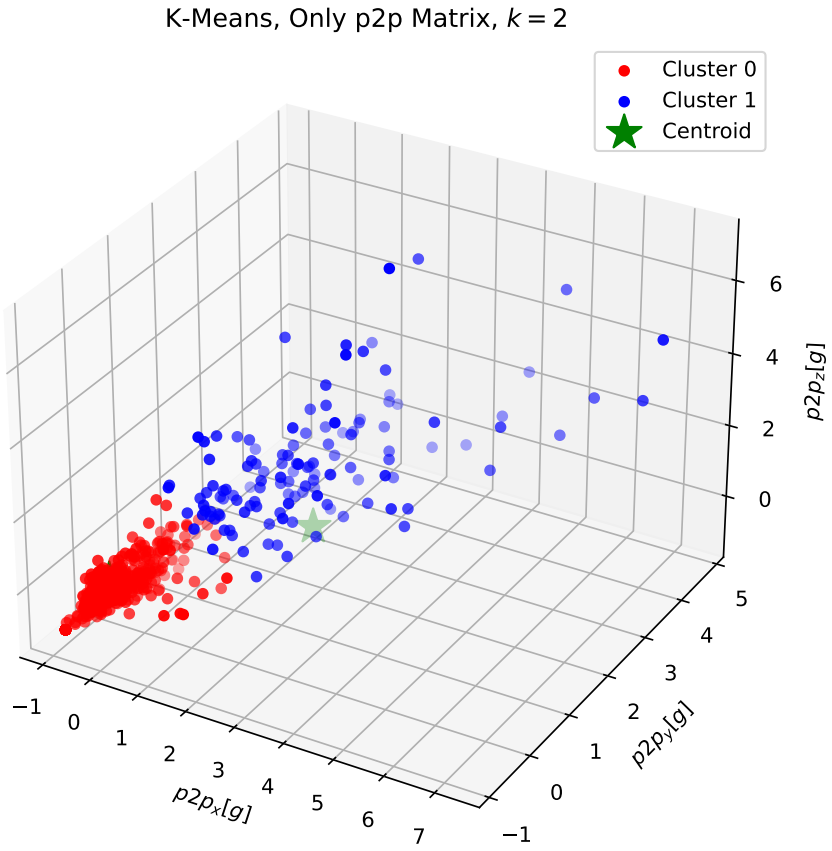


Figure 7.2: Result of the K-Means algorithm when the input is the *p2p* matrix and the number of cluster is $k = 2$.

incorrectly identifies smooth road instances as irregularities) and 22 instances classified as "false negatives" (cases where irregularities are inaccurately overlooked by the algorithm).

7.1.2 Only *std* Matrix

The analysis continued by utilizing the *std* matrix as the input for the algorithm. The code for training the algorithm closely resembles the previously presented one, with the distinction that the input for the *KMeans*() classifier is now the *std* matrix, incorporating the standard deviation values of the three accelerations. The model parameters remain consistent with those employed in the prior analysis, with *init* set to *random* and *n_init* = 10. The result is displayed in the Fig. 7.3. The model trained with this configuration successfully clusters over 99% of the windows associated with smooth road conditions, and achieve an accuracy of 91% in classifying windows that represent irregularities. There are a total of 4 instances classified as "false positives", and 13 instances classified as "false negatives".

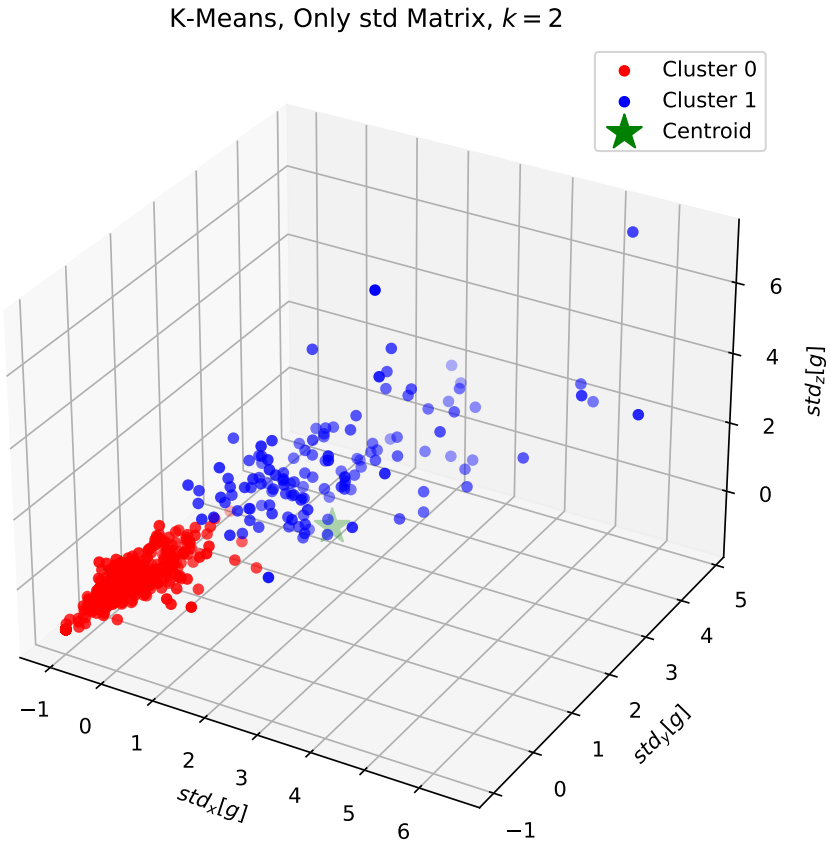


Figure 7.3: Result of the K-Means algorithm when the input is the *std* matrix and the number of cluster is $k = 2$.

7.1.3 *p2p* and *std* Matrices Combined

The input matrix now is the *combined* matrix defined earlier. The code and the parameters to train the algorithm remained once again the same as the previous analysis, and the results are shown in Fig. 7.4. It's noteworthy that the algorithm takes six measurements as input, three for the *P2P* value of each acceleration and three for the standard deviation of the same accelerations. However, for the purpose of visualization, the points are plotted in a three-dimensional space, utilizing the x, y, and z axes to represent the $P2P_x$, $P2P_y$, and $P2P_z$ values, respectively. It's important to mention that utilizing the standard deviation as axes for plotting would yield equivalent results. The model achieved an accuracy of approximately 95% in classifying smooth road segments and 90% in classifying irregularity, with a total of 3 false negatives and 14 false positives.

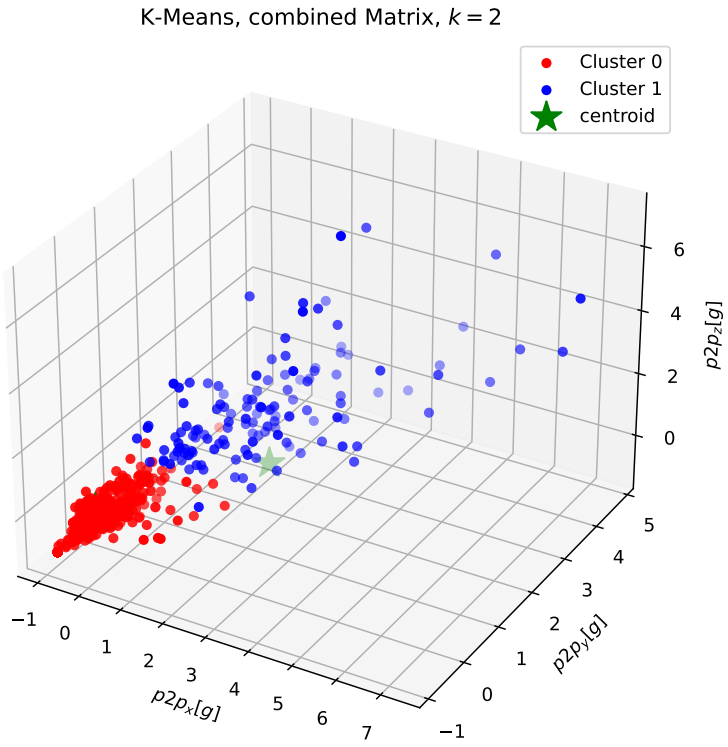


Figure 7.4: Result of the K-Means algorithm when the input is the *combined* matrix and the number of cluster is $k = 2$.

7.2 Multiclass Classification

Similarly to the studies carried out before, the K-Means classifier is also employed to distinguish between smooth road, low severity irregularity and high severity irregularity. Fig. 7.5 displays the accurate classification when the labels are assigned as follows: 0 for smooth road, 1 for low-impact irregularity, and 2 for irregularities of high severity. The red dots are associated to smooth road instances, blue and green dots are respectively associated with low and high severity irregularity instances. The *KMeans*() function is initiated with the same parameters as before, with the only difference being the number of cluster *n_cluster* now equal to 3.

7.2.1 Only *p2p* Matrix

First, the *p2p* matrix is selected to be the input to the algorithm. The results are shown in Fig. 7.6. The plot makes it evident that distinguishing the severity of irregularities poses a more challenging task. Various factors contribute to this complexity, including the speed at which the vehicle encounters the irregularity. For instance, Fig. 7.7 illustrates two runs over the same irregularity but at different speeds. The upper plot depicts the three-axis accelerations when the vehicle tra-

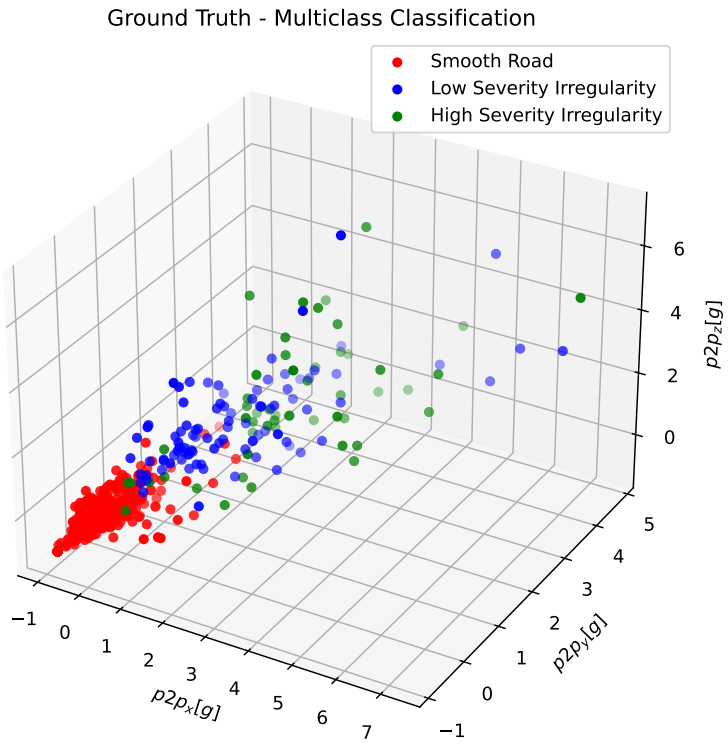


Figure 7.5: Ground truth with the *p2p* matrix as input and $k = 3$.

verses the bump at a speed of approximately 10km/h. Subsequently, the same vehicle navigates the same bump at a speed of approximately 15km/h. The plot distinctly captures a varied traces of the accelerations. Notably, higher speeds induce more vehicular shake, consequently augmenting the *P2P* measure of the window where the irregularity is located. However, once again, the algorithm exhibits good performances in recognizing smooth road, achieving an accuracy of 98%. A fairly high results is evident also in recognizing low severity irregularity, with an accuracy of 85%, while the major problems are in clustering high severity irregularity correctly, reaching an accuracy of 52%. These results are summarized in the confusion matrix displayed in Fig. 7.8.

7.2.2 Only *std* Matrix

The input of the classifier is now the *std* matrix. The results are shown in Fig. 7.9. The same considerations as the previous case can be drawn. In particular, the algorithm when trained with this configuration exhibits an accuracy of above 99% for the classification of smooth road segments, 83% regarding low severity irregularity and only 37% accurate when classifying high severity irregularity. The results are once again summarized in the confusion matrix displayed in the Fig. 7.10.

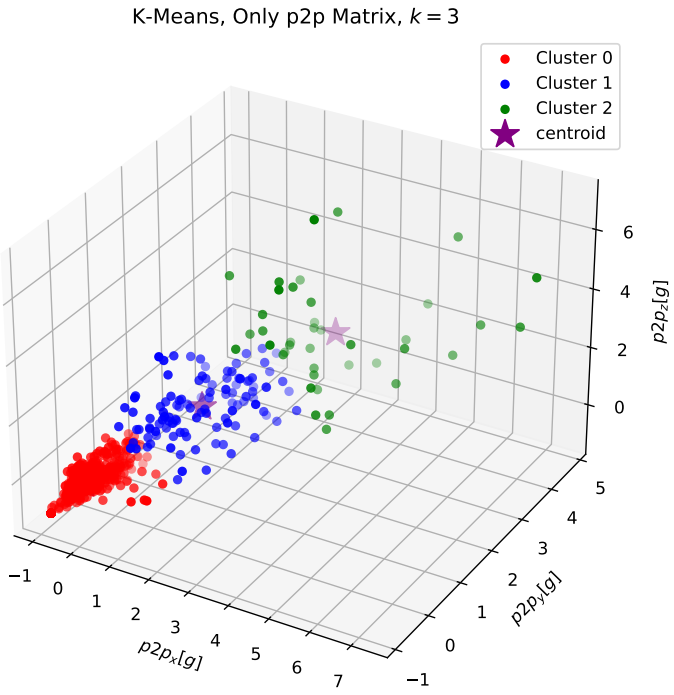
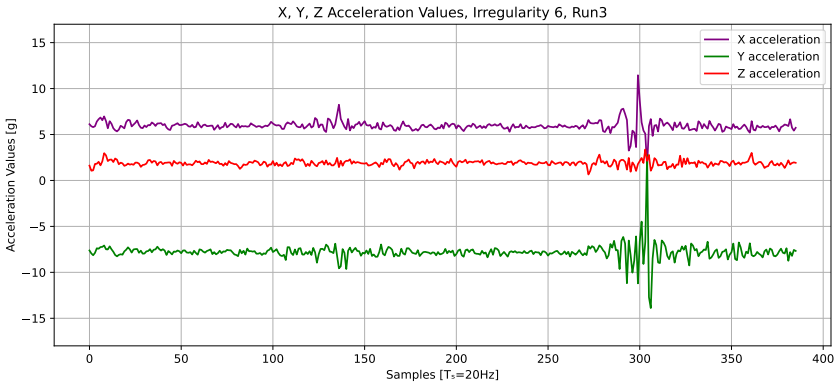


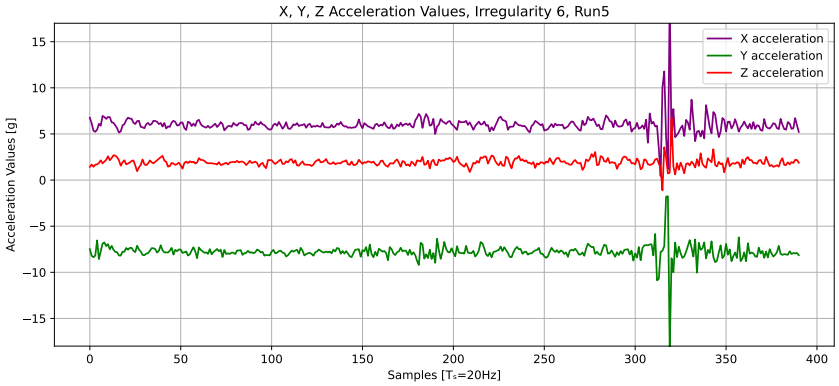
Figure 7.6: Result of the K-Means algorithm when the input is the *p2p* matrix and the number of cluster is $k = 3$.

7.2.3 *p2p* and *std* Matrices Combined

The analysis ended testing the model with the *combined* matrix as input and the number of initial clusters equal to three. The results are shown in the Fig. 7.11. Looking at the confusion matrix plotted in Fig. 7.12, it is evident that, once again, the model has no problem in clustering together instances corresponding to smooth road and to low severity irregularity. The difficulty are found when trying to separate samples corresponding to high severity irregularity.



(a) *Irregularity* 6, *Run* 3



(b) *Irregularity* 6, *Run* 5

Figure 7.7: Two different runs over the same irregularity at different speeds.

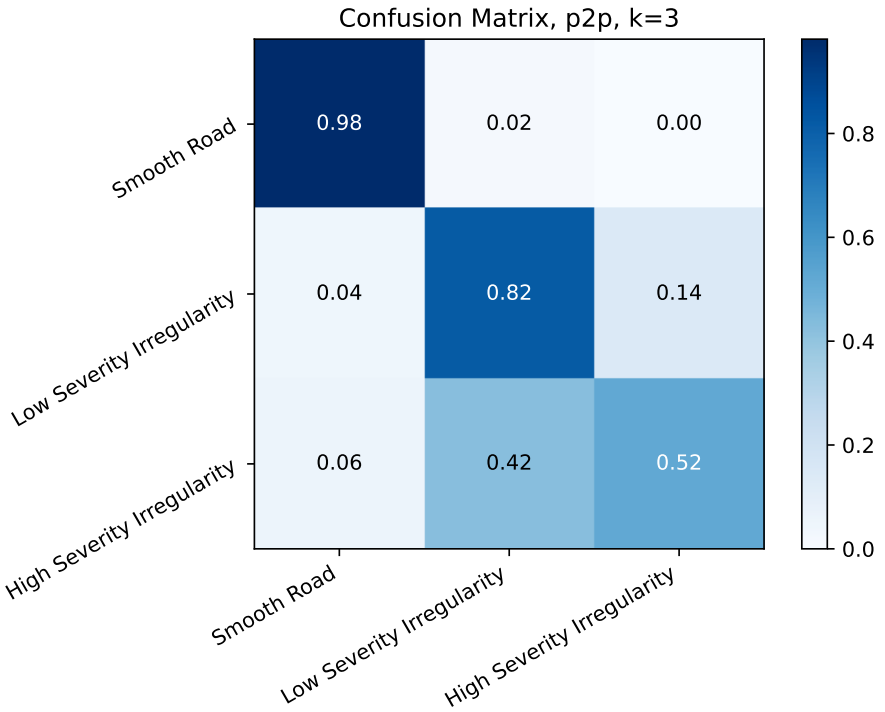


Figure 7.8: Confusion matrix when the input is the *p2p* matrix and the number of clusters is $k = 3$.

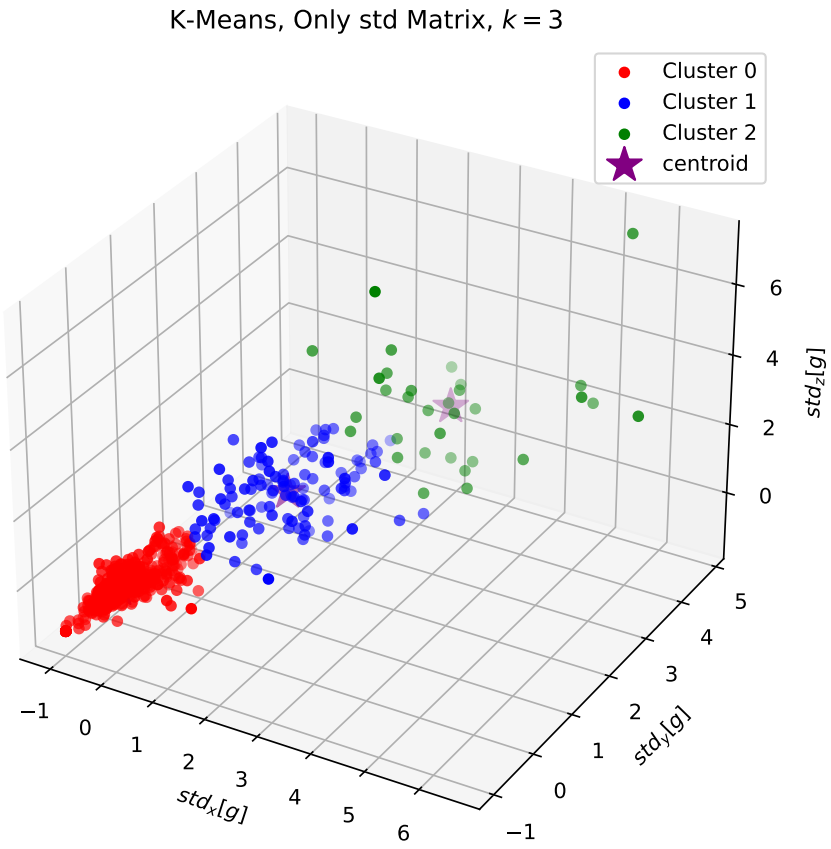


Figure 7.9: Result of the K-Means algorithm when the input is the *std* matrix and the number of cluster is $k = 3$.

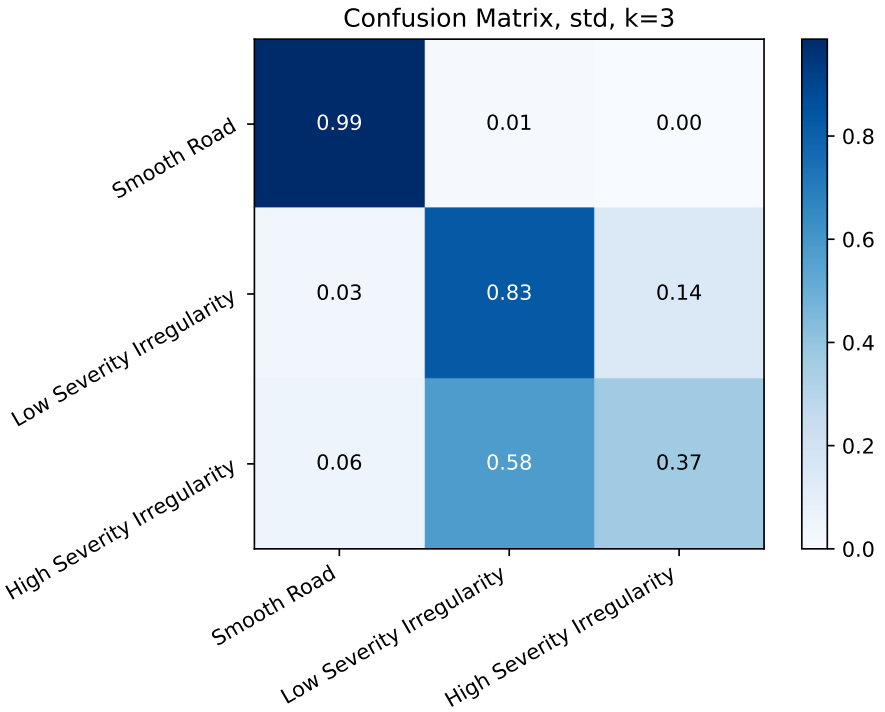


Figure 7.10: Confusion matrix when the input is the *std* matrix and the number of clusters is $k = 3$.

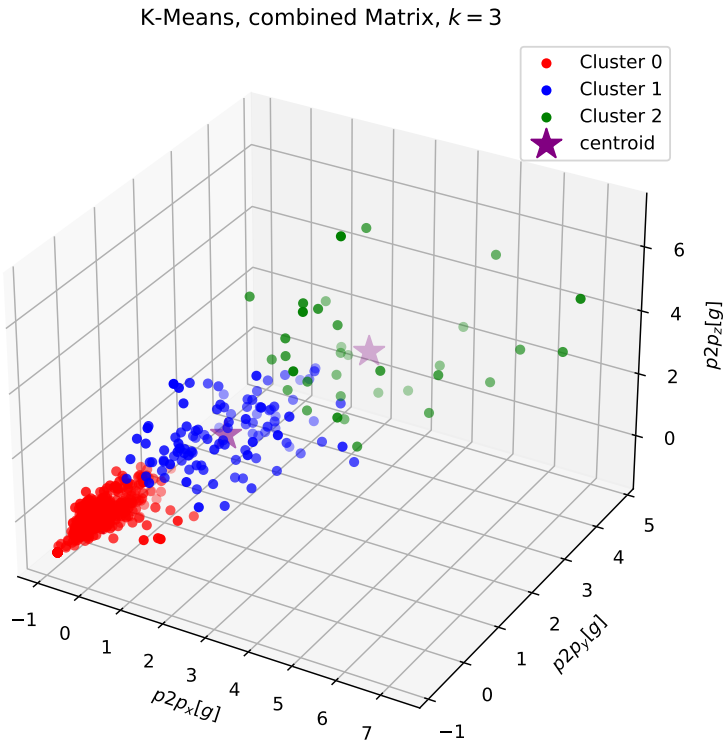


Figure 7.11: Result of the K-Means algorithm when the input is the *combined* matrix and the number of cluster is $k = 3$.

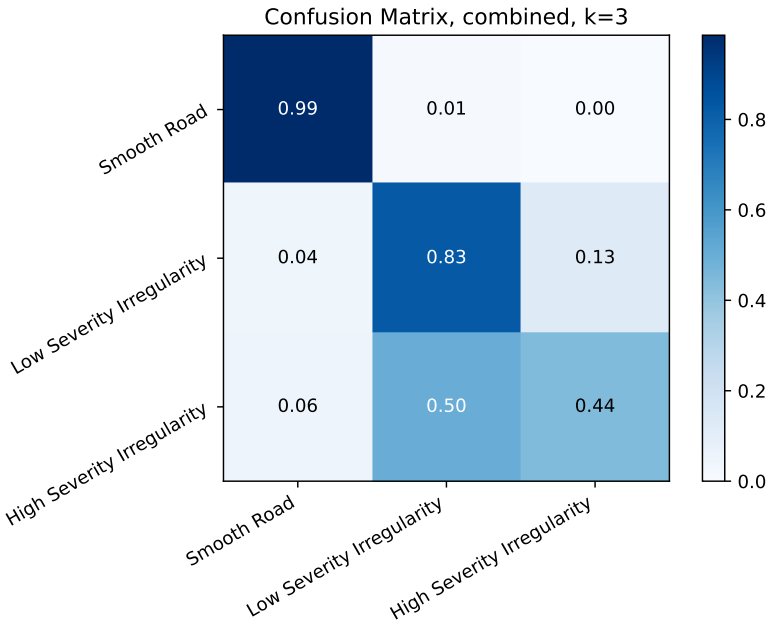


Figure 7.12: Confusion matrix when the input is the *combined* matrix and the number of clusters is $k = 3$.

8 Conclusions and Future Work

In this concluding chapter of the thesis, we discuss the key findings and insights derived from the research. Additionally, potential routes for future work are presented, providing a roadmap for further exploration and development in the field.

8.1 Summary of the Results

In this section, we compare the optimal results achieved for each of the considered classifiers. The observations are categorized into binary and multiclass classification, and commented based on the best outcomes obtained in both analyses. Recall that the whole dataset comprises 800 instances in total. For both supervised machine learning techniques, the test set is set to be 30% of the whole dataset, thus containing 240 samples. Regarding the K-Means classifier, all the dataset is considered during the test phase.

8.1.1 Binary Classification

In the context of binary classification, the best results obtained with the three different classifier are summarized in the Table 8.1. For binary classification, the algorithm exhibiting the highest overall performance is RF when utilizing the *p2p* matrix as input, achieving a remarkable test accuracy of 100%. This result should be taken with a grain of caution, as the algorithm may experience a slight decrease in accuracy when the input is derived from less stable conditions. It is noteworthy, however, that across various configurations tested with different algorithms, consistently high results were obtained. This suggests that, in the context of binary classification, all the algorithms examined demonstrate good accuracy even when presented with diverse datasets. It is important to highlight also the good performances achieved by the K-Means classifier. Considering its unsupervised nature, it is likely that the algorithm may encounter more challenges than

Table 8.1: Summary of the results obtained for binary classification with the different configurations.

Classifier	Input	Accuracy	Training Error	Test Error	False Positives	False Negatives
SVM	p2p	0.9678	0.034	0.0041	0	1
	std	0.983	0.0196	0.0083	2	0
	combined	0.9875	0.0107	0.0042	1	0
RF	p2p	1	0.0268	0.0	0	0
	std	0.9875	0.0196	0.0124	3	0
	combined	0.9875	0.0179	0.0125	3	0
KMeans	p2p	0.965	N/A	N/A	6	22
	std	0.9788	N/A	N/A	4	13
	combined	0.9788	N/A	N/A	3	14

Table 8.2: Summary of the results obtained for binary classification with the different configurations.

Classifier	Input	Accuracy	Training Error	Test Error	$0 \rightarrow 1$	$0 \rightarrow 2$	$1 \rightarrow 0$	$1 \rightarrow 2$	$2 \rightarrow 0$	$2 \rightarrow 1$
SVM	p2p	0.9196	0.0411	0.0917	0	0	2	7	0	13
	std	0.9321	0.066	0.071	1	0	0	3	1	3
	combined	0.9357	0.034	0.075	2	0	1	7	1	7
RF	p2p	0.933	0.0553	0.0667	0	0	1	6	0	9
	std	0.9166	0.048	0.083	2	0	0	9	0	9
	combined	0.9291	0.035	0.071	2	0	0	7	0	8
KMeans	p2p	0.9325	N/A	N/A	12	0	4	13	3	27
	std	0.9288	N/A	N/A	8	0	3	13	3	19
	combined	0.9325	N/A	N/A	9	0	4	12	3	26

its supervised counterparts. The fairly high results obtained by K-Means are attributed to the high discriminative capability of the selected features, which are hence suitable for the classification task in hand.

8.1.2 Multicalss Classification

The best results obtained for multiclass classification are summarized in the Table 8.2. The six right-most columns of the table show the instances that were misclassified. The label of each column indicated the type of misclassification in the form $X \rightarrow Y$, where X is the actual road class the sample belongs to and Y is the one inferred by the algorithm. The numbers in each column represent the count of the misclassified road samples. The results confirm the increased difficulty in distinguishing between two types of irregularities, as previously discussed in the preceding chapters. Notably, the SVM classifier demonstrates fairly high performance, particularly when

the input is the *combined* matrix. Once again, the K-Means classifier emerges as the algorithm encountering more challenges than its counterparts. This outcome aligns with expectations, considering that the model relies solely on the distance measure between points in the feature space. As explained in the dedicated chapter, various factors, including the speed of the vehicle traversing the irregularity, can influence the accelerometer's trace. A possible improvement in the model can be incorporating the speed of the vehicle into the prediction process, assigning distinct weights to each instance accordingly.

8.2 Conclusions

This thesis has showcased the feasibility of predicting road pavement conditions using low-cost sensors integrated into the vehicle. The absence of direct involvement required from end-users represents a significant stride towards advancing safety for Vulnerable Road Users (VRUs).

8.3 Possible Follow-Up

As mentioned in the introduction, the goal of this thesis is to set the ground for a fully functional driver assistance to improve the safety of VRUs. Numerous potential extensions to this work exist, beginning with the identification of specific types of irregularities such as potholes, bumps, transverse cracks, and more. Furthermore, through this research, it becomes conceivable to construct an always up-to-date map of the city using algorithms trained for irregularity recognition. Subsequent research endeavors can also explore the feasibility of real-time application of these methods and the mechanisms for transmitting data to the computational unit responsible for processing the information. Addressing the challenges associated with soft mobility necessitates grappling with the issue of information exchange. The nature of the data collected suggests that on-site processing might not be feasible. Consequently, the need to transmit data to an external facility adds an extra layer of complexity to the overall system.

References

[1] P. Olszewski, P. Szagata, D. Rabczenko, and A. Zielinska, "Investing safety of vulnerable road users in selected eu countries," *Journal of Safety Research*, 2019.

[2] R. V. Belavkin, P. M. Pardalos, J. C. Principe, and R. L. Stratonovich, "Definition of the value of information," *Theory of Information and its Value*, 2020.

[3] D. Merdrignac, P. Shagdar, O. Jemaa, and F. Nashashibi, "Study on perception and communication systems for safety of vulnerable road users," *IEEE 18th International Conference in Intelligent Transportation Systems*, 2015.

[4] D. Merdrignac, P. Shagdar, and F. Nashashibi, "Fusion of perception and v2p commiunication systems for the safety of vulnerable road users," *IEEE Transaction on Intelligent Transportation Systems*, vol. 18, no. 7, 2017.

[5] P. Sewalkar and S. Jochen, "Vehicle-to-pedestrian communication for vulnerable road users: Survey, design considerations, and challenges," *Sensors*, vol. 19, 2019.

[6] Y. Wu, M. Abdel-Aty, O. Zheng, Q. Cai, and L. Yue, "Developing a crash warning system for the bike lane area at intersection with connected vehicle technology," *Transportation Research Record*, vol. 2673, 2019.

[7] A. Al-Shaghouri, R. Alkhatib, and S. Berjaoui, "Real time pothole detection using machine learning," 2010.

[8] M. H. Asad, S. Khaliq, M. H. Yousaf, M. O. Ullah, and A. Ahmad, "Pothole detection using machine learning: A real-time and ai-on-the-edge perspective," *Advances in Civil Engineering*, 2022.

[9] A. Kumar, Chakrapani, D. J. Kalita, and V. P. Singh, "A modern pothole detection technique using deep learning," *IEEE Xplore*, 2022.

[10] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: Rich monitoring of road and traffic conditions using mobile smartphones," *In Proceeding of the 6th ACM Conference of Embedded Network Sensor Systems*, pp. 357–358, 2008.

[11] N. Sabir, A. A. Memon, and F. K. Shaikh, "Threshold based efcient road monitoring system using crowdsourcing approach," *Wireless Personal Communications*, 2019.

[12] S. Adwan and H. Arof, "On improving dynamic time warping for pattern matching," *Measurement*, vol. 45, pp. 1609–1620, 2012.

[13] P. Senin, "Dynamic time warping algorithm review," *Information and Computer Science Department University of Hawaii Manoa: Honolulu*, 2008.

[14] C. Wu, Z. Wang, S. Hu, *et al.*, "An automated machine-learning approach for road pothole detection using smartphone sensor data," *Sensors*, vol. 20, no. 5564, 2020.

[15] A. Basavaraju, J. Du, F. Zhou, and J. Ji, "A machine learning approach to road surface anomaly assessment using smartphone sensors," *IEEE Sensors Jurnal*, vol. 20, no. 5, 2020.

[16] M. Perttunen, O. Mazhelis, F. Cong, *et al.*, "Distributed road surface condition monitoring using mobile phones,"

[17] K. Pawar, J. Siddhi, and B. Smita, "Efficient pothole detection using smartphone sensors," *ITM Web of Conferences*, vol. 32, 2020.

[18] Arduino. "Arduino mkr wi-fi 1010." (), [Online]. Available: https://docs.arduino.cc/resources/datasheets/ABX00023-datasheet.pdf.

[19] Arduino. "Wifinina." (), [Online]. Available: https://www.arduino.cc/reference/en/libraries/wifinina/.

[20] R. Schwarz. "Understanding uart." (), [Online]. Available: https://www.rohde-schwarz.com/us/products/test-and-measurement/essentials-test-equipment/digital-oscilloscopes/understanding-uart_254524.html#:~:text=UART%20stands%20for%20universal%20asynchronous,also%20have%20a%20ground%20connection..

[21] C. Basics. "Basics of the i2c communication protocol." (), [Online]. Available: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/#:~:text=I2C%20is%20a%20serial%20communication,always%20controlled%20by%20the%20master..