UNIVERSITÀ DEGLI STUDI DI BERGAMO

Department of Management, Information and Production Engineering

Master's Degree Thesis in Computer Engineering

Class n. LM-32 - Class of Master's Degrees in Computer Engineering

Vibration-based Terrain Recognition for E-Bikes Using Inertial Sensor Measurements

Supervisor:
Prof. Mirko Mazzoleni

Co-supervisor:
Dott. Nicholas Valceschini

Author:
Cristiano Gambirasio
Matr. 1066866

Academic Year 2023/2024

Preface

This paper presents the result of the research conducted as part of my master's thesis in Computer Engineering, marking the culmination of my academic journey at the University of Bergamo. The focus of this work is on addressing the terrain recognition task for an e-bike using a vibration-based approach.

I would like to express my gratitude to the faculty, and in particular the Control Systems and Automation Laboratory for giving me the opportunity to contribute to this research by providing me with the e-bike prototype that is the subject of this study, as well as the necessary tools for the project.

I am especially grateful to my supervisor, Mirko Mazzoleni, for his support, mentorship, and the time he dedicated to me during these months. I would also like to thank my co-supervisor, Nicholas Valceschini, for his feedback and insights, which were essential for the development of this work.

Abstract

One of the main challenges faced in urban areas is enhancing the mobility of people to make it as efficient, fast, and environmentally sustainable as possible. A key component in achieving this goal is promoting active mobility which can be defined as transportation methods that are low-cost, zero-emissions, and also offer physical benefits by supporting a more active lifestyle. One of the most widely diffused active ways of moving nowadays, besides walking, is cycling. In order to promote the use of the latter, modern, safe, and comfortable bikes should be studied.

One of the possible features that such a modern lightweight vehicle should have is terrain recognition, defined as the ability to detect the type of terrain the vehicle is traversing based on sensor data. In this study this functionality has been implemented on a sensor-equipped e-bike prototype, enabling the distinction between three terrains: asphalt, gravel, and cobblestone. The process to achieve this objective began with the collection of sensor measurements on the three terrains in order to create a labeled dataset. Once collected, the data was cleaned and then analyzed in both the time and frequency domains to extract relevant features for terrain differentiation. Finally, a machine learning algorithm capable of determining terrains based on the measurements observed by the sensor has been implemented. This model was trained and optimized to improve accuracy in terrain recognition, allowing reliable real-time predictions.

The methodology presented in this study can be followed as a guideline to implement new terrain recognition algorithms using inertial measurements or to improve this work by adding terrains into the current model. The practical result of this work is an algorithm able to recognize the three terrains on which the model is trained.

Contents

Chapter 1

Introduction

1.1 Context

Active mobility refers to the transportation of individuals using zero-emission, low-cost means, such as walking and cycling. This method of travel is particularly convenient for urban environments, as it offers a level of independence that is often absent in conventional public transportation systems, while simultaneously helping to mitigate issues commonly associated with car usage, such as pollution and increased traffic congestion in urban areas.
The advantages of active mobility are also noteworthy from a governmental perspective as it represents a sustainable method of transport which at the same time allows an increase in the population's health metrics. Active mobility fosters healthier living habits, reducing the prevalence of lifestyle-related health issues such as obesity, diabetes, and cardiovascular diseases, which are often linked to sedentary behavior. For this reason, it's in the government's interests to encourage the use of these means of transport with policies, infrastructure, and vehicle development that make sustainable mobility more attractive to people.

Regarding this topic, the European Union has strongly promoted active mobility in the EU Urban Mobility Framework [1], designating 431 cities as urban nodes and setting specific requirements for them. One of these is to adopt a Sustainable Urban Mobility Planning (SUMP). The latter is a strategic plan that should be designed to satisfy the transportation needs of the entire urban area, whose principles are based on sustainable and active transport. This initiative is further encouraged by declaring that EU funding for urban mobility projects should be highly linked with having a SUMP.
About European funding, in 2020, the Union Council allocated 750 billion euros for the recovery of nations after the COVID-19 pandemic. Thanks to this, Italy in July 2021 approved the PNRR, which has the target of developing digitalization and green solutions. The second component of the fifth mission stipulated in this plan includes the creation of five national centers that is, a network of universities, institutions, and enterprises committed to research, develop, and implement solutions that are immediately usable and highly scalable for the entire social context. One of these is Centro Nazionale MOST [2], the Italian Sustainable Mobility Center that

manages the collaboration of 24 universities, the Italian National Research Council, and 24 companies dealing with finding modern and sustainable solutions for public transport. This National Center is organized with the "Hub&Spoke" model, so 14 Spoke Leaders develop research programs on different transport fields.
The fifth spoke, led by the University of Bergamo, focuses on active mobility and lightweight vehicles, on making them safer, and also on the creation of a smart infrastructure to have updated information on road conditions.

1.2 Work Goal

This thesis project is part of the efforts to develop and evolve lightweight vehicles. In particular, the study focuses on an e-bike prototype with the aim of using the measurements obtained from the vehicle's sensors to implement and optimize a terrain recognition algorithm. This task involves the classification of the type of terrain on which a vehicle is currently traveling.

The primary goal of this software is to generate real-time predictions regarding the nature of the ground beneath the vehicle, providing an accurate classification output. Secondly, stability in terrain classification is of relevant importance, since frequent fluctuations in the predicted terrain type would undermine the effectiveness and usability of the system, potentially leading to confusion and worsening the user experience.

Such an algorithm could be useful for the objective of the fifth MOST spoke. For example, by enabling continuous monitoring of the terrain, an adaptive motor response can be developed optimizing performance, safety, and comfort. Additionally, the algorithm could facilitate the sharing of terrain data with other systems, contributing to a more connected and responsive transportation network.

1.3 Thesis Outline

The approach adopted in this thesis involves a comprehensive review of the existing literature related to the problem. After evaluating various approaches and methodologies, the most suitable one for this specific case will be selected and implemented, with adjustments made where necessary to adapt it to the requirements of the specific case.
All analyses and implementations will be conducted using MATLAB 2024a.

The thesis will describe this process following the outline below:

- **Chapter 2, State of The Art**: Provides a comprehensive overview of the current state of the art in terrain recognition. In particular, the choice of the sensors, the feature extraction techniques, and the typically used classifier will be studied. This research will represent the literature baseline for this project.
- **Chapter 3, Data Acquisition**: Describes the hardware components used to

capture the terrain data and provides a thorough examination of the specifications of the hardware setup. Additionally, this chapter presents a step-by-step description of the data collection procedures, emphasizing the operational protocols followed to ensure consistent and high-quality data acquisition.

- **Chapter 4, Data Analysis and Preprocessing**: Contains visualization, cleaning, analysis, and preprocessing of the collected dataset. The chapter will cover all the steps performed to go from the raw dataset collected from the sensor system to a set of meaningful features to be fed as input to the classification model.

- **Chapter 5, Terrain Recognition Algorithm**: This chapter starts with some preliminary premises on the machine learning techniques used, followed by an accurate description of the offline model training and tuning procedure. Finally, an architecture for stable real-time predictions is presented and evaluated.

- **Chapter 6, Conclusions**: The results achieved by this project are presented and discussed. After that, a list of future works to improve and expand the algorithm is suggested.

- **Appendix A, Theoretical Background**: This appendix provides a detailed theoretical background for the machine learning algorithms that have been implemented as part of this work. It aims to establish a solid understanding of the fundamental principles, mathematical foundations, and key concepts underlying these machine learning techniques.

Chapter 2

State of the Art

Most of the researches regarding terrain recognition are done in the robotic and autonomous vehicle field, however, these methodologies with some adaptations can be useful as a guideline for this project.

2.1 Data-driven and Model-based approaches

Terrain recognition is a classification problem operating on a physical system, the e-bike. The solutions to this task can be divided into two different approaches [3].

The data-driven approach consists of models that are trained using historical data. The idea is that the system "learns" from this data by recognizing patterns, correlations, and trends, without the need to be explicitly programmed with a set of rules. This approach is usually divided into a phase of offline learning where a loss function is minimized to fit the training dataset, and the online classification of new samples based on the model learned offline.

In model-based systems, the laws of physics are used as the basis for creating specific rules. These models describe a physical system through precise mathematical laws that are established by analyzing the latter and modeling its components. Sometimes this technique use data to estimate parameters of the equations, but always assuming that the system follows the equations established.

Comparing these two approaches, the data-driven one offers more flexibility because it is not based on any mathematical model of the physical system allowing it to adapt even to uncertainties of this, but it requires an extensive phase of data collection to perform the training of the model. On the other hand, the model-based approach allows the inclusion of prior knowledge in the form of equations describing the behavior of the system, reducing effort in the collection of data. However, because of its assumptions on the system, the results obtained are limited to known kinds of models, and extensions to more general scenarios are very challenging.

To solve the terrain recognition task, therefore, two different approaches could be followed:

1. **Data-driven:** classifying the terrains by signal recognition of labeled data;
2. **Model-driven:** using a solid mechanics model to analytically predict how the bike will respond to the interaction with distinct terrains;

The approach that this work will follow is the first one since it is the most used in literature and provides good results in the analyzed papers. Furthermore, as observed by Sadhukhan while solving the terrain recognition task for a military experimental unmanned vehicle, the accuracy achieved using pattern recognition techniques improves while the speed increases [4]. While the autonomous vehicle studied in this paper reach a maximum velocity of about $13\frac{km}{h}$, a bike ride usually experience an higher mean speed, so a data-driven approach seems a valid choice.

2.2 Sensor choice

To solve the terrain recognition task, the sensor choice is mainly divided into two categories: inertial sensors and vision systems.

Inertial sensors

Inertial sensors are devices that measure an object's acceleration, angular rate, and sometimes orientation.
A first set of research [5][6][7][8], chose an accelerometer on one or three axes as a sensor. When only an axis is considered, the studies observe the perpendicular to terrain one, since it contains the most information regarding the ground-wheel interaction. Another parameter that has to be decided is the sampling rate, that is, the frequency with which measurements are collected. Although some work uses accelerometers with frequencies up to 4 kHz, it is demonstrated that a 100 Hz sensor is sufficient for this task [5][8].
A second approach to vibration sensing is described by Brooks and Iagnemma, mounting a contact microphone to a vehicle wheel [9]. However, although this device is a valid option for low-speed rovers, it's not suitable for detecting high-intensity vibrations such as those of a moving bike.
The vibrations could also be measured with an IMU [10]. IMU stands for Inertial Measurement Unit, and it's the composition of three accelerometers, three gyroscopes, and occasionally three magnetometers. While the papers analyzed until now only rely on the accelerometer measurements, Tick's paper shows that some information regarding the terrain beneath the bike could be in its angular velocity so the use of an IMU could enhance the prediction performance.

Vision systems

A completely different way to solve this task is by training a computer vision system capable of distinguishing terrains based on their texture [11]. Vision-based classi-

fication generally has good results but also introduces problems that need to be considered, for example:

- Sensitivity to changes in lighting conditions such as shadows, glare, or uneven lighting, which can distort the features of surfaces, leading to decreased accuracy in classification tasks. This is especially problematic in real-world environments where lighting is rarely consistent.
- Difficulty in recognizing the load-bearing surface. These systems typically rely on analyzing the road's most superficial layer, which can lead to inaccuracies, for example in environments covered with vegetation.
- A camera pointing at the ground is subject to dust, mud, or other debris that can accumulate on the camera lens, drastically reducing the quality of the captured image. This can make accurate classification impossible, as the obstruction of the view compromises the system's ability to properly analyze the underlying terrain.

For this work, a focus on inertial measurements as features for terrain prediction has been preferred because of the reliability of this approach and the better interpretability that can be achieved.

2.3 Feature extraction

After the measurement's collection, the next step is extracting from these meaningful values and metrics capable of characterizing the different classes. This procedure usually can be done in two different ways.

Time domain

The first method is by looking at statistics in a specific interval in the time domain [5][6][7]. As an example on a set of n observations $\{x_1, x_2, ..., x_n\}$:

- Mean value ($\bar{x}$):
$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$
- Variance (σ^2):
$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$
- Threshold crossings (th_{cross}): Number of times a threshold is crossed. Examples of thresholds are:
  - 0
  - $\bar{x}$
  - Manually selected thresholds

- Maximum value ($x_{\max}$):
$$x_{\max} = \max_{1 \leq i \leq n} x_i$$
- Minimum value ($x_{\min}$):
$$x_{\min} = \min_{1 \leq i \leq n} x_i$$
- Peak-to-peak value (x_{pp}):
$$x_{pp} = x_{\max} - x_{\min}$$

Frequency domain

Then, some other information can be found by looking at the signal's frequency spectrum. To do this, the FFT (Fast Fourier Transform) or PSD (Power Spectral Density) are calculated.
Some of the analyzed papers extract features by computing statistics in the frequency domain [6][7].
Examples of the features extracted by a frequency analysis with F points (with amplitude $\{A_{f_1}, A_{f_2}, ..., A_{f_F}\}$ on the frequencies $\{f_1, f_2, ..., f_F\}$) performed on an interval of n observations $\{x_1, x_2, ..., x_n\}$ are:

- Amplitude mean ($\bar{A}$):
$$\bar{A} = \frac{1}{F} \sum_{i=1}^{F} A_{f_i}$$
- Sum of higher half of amplitude spectrum ($A_{MaxHalf}$):
$$A_{MaxHalf} = \sum_{i=1}^{F/2} A_{f_i} \quad | \quad A_{f_i} \geq \bar{A}$$
- Centroid of the amplitude spectrum (f_C):
$$f_C = \frac{\sum_{i=1}^{F} f_i A_{f_i}}{\sum_{i=1}^{F} A_{f_i}}$$
- Frequency variance (f_v):
$$f_v = \frac{\sum_{f=1}^{F} (f_i - f_C)^2 A_{f_i}}{\sum_{i=1}^{F} A_{f_i}}$$

Other researchers prefer to use the entire computed spectrum as features for the prediction [8]. This approach makes feature dimensionality explode, and so is suitable only in models capable of handling a high number of features. A way to solve this problem is to reduce feature complexity using PCA (Principal Component Analysis) [9].

Finally, Tick's paper [10] shows the results of an automatic feature extraction technique called "OpenSMILE", initially thought for audio applications, but also useful for generic vibration signals. This software can compute and extract an enormous amount of features, so it also needs an effort in feature selection.

2.4 Machine Learning Classifiers

A classification algorithm must be selected to enable the prediction of a class based on the extracted features.

The classifiers found during the literature analysis can be divided into classic machine-learning algorithms and deep-learning techniques. The main differences between these groups of algorithms are two.
The first is the kind of input with which the algorithm is fed, while for classical machine learning methods, an accurate procedure for feature extraction has to be followed, using a neural network the input can be less elaborate because the model will find meaningful patterns automatically.
This functionality introduces the second difference, in fact, the automatic extraction of features performed by deep-learning models will make these produce a less interpretable output because of their black-box nature.

The traditional machine-learning approaches encountered are:

- **Linear Classifier:** Proposed in Brooks paper [9]. To extend this model to a multi-class situation a set of classifiers combined with a voting system has been ideated. Besides being the simplest algorithm in literature, the results are already somehow satisfactory.
- **Linear Bayes Classifier:** Another algorithm that can be used as a classifier is a Bayesian one [10], which is based on finding the class that maximizes the a posteriori probability using the MAP (Maximization of A Posteriori) rule. The conditional probability density function can be estimated non-parametrically, but it could be hard with a high number of features. To solve this problem usually, a normal distribution of data is assumed, so that a parametric estimation of the CPDF can be done.
- **SVM:** The most used technique in the analyzed literature is the Support Vector Machine [5][7] which will be further explained in appendix A.1.

In contrast with these methods, some deep-learning algorithms were also found during the research. These techniques are based on Neural Networks that is, structures based on layers of interconnected nodes, or "neurons", each of which performs a simple computation. Two mainly used NN architectures have been found:

- **DNN:**[6][8] A Deep Neural Network refers to a network that contains more hidden layers between the input and the output stages. For example in Giguere's work [6] eight input nodes (one for every feature) and ten output nodes (one for every class) are connected via two hidden layers with twenty neurons each.
- **RNN:**[12][13] A Recurrent Neural Network (RNN) is a type of neural network designed for processing sequences of data, such as time series. Unlike traditional neural networks, which process input data independently, RNNs have connections that allow information to persist across time steps which could be useful for terrain recognition.

Comparing the results of all the analyzed papers, these do not highlight any particular differences in performance between classical machine learning and deep learning models. During this project classical machine learning has been preferred since it require both less training time and less training data [14], and so it's more appropriate for small and specific tasks like terrain recognition. In addition to this it also ensures better interpretability of the model and prediction explainability.

Chapter 3

Data Acquisition

This chapter will describe the procedures and the hardware used during the dataset creation.

3.1 Hardware configuration

The hardware available for this project is a sensor-equipped e-bike prototype, part of the MOST Spoke 5 project.

3.1.1 E-Bike

The e-bike is an OLMO E.KOLT (figure 3.1), which is an electric mountain bike, so it has a front fork for shock absorption that can be manually locked.
The motor mounted for assisted pedaling is a Polini EP3+ MX, capable of delivering 90Nm of torque and is powered by a Portapower battery with a 711 Wh capacity.

Figure 3.1: The e-bike used in this work (on the right)

Using a controller (figure 3.2) is possible to choose between six levels of electric assistance, from none at level 0 to maximum at level 5. The other two buttons are used for navigation in the Polini motor control unit, and for switching an optional headlamp. The functionality of the latter has been changed, allowing the control of a new data acquisition device described in section 3.1.2.
The e-bike already has a sensing system that provides information about battery

status, assistance level, current speed, and cadence that are not usually available, but the manufacturer had offered to give open access to these for this research scope.

Figure 3.2: E-bike motor controller

3.1.2 BRICK

The sensorization of the e-bike has been implemented by the electronics laboratory of the Engineering and Applied Sciences Department at the University of Bergamo. The device developed, called "BRICK", is a set of sensors connected to a Raspberry Pi 3B+, a low-cost single-board computer.

DAM

The Raspberry Pi has been used as a data acquisition module (DAM), and it is a key component serving as the central unit for the whole sensing system. This module is responsible for collecting and storing data from the other sensor modules and enables communication between the BRICK and the user. The choice of the board is explained by the features it offers despite the low cost, like a wide choice of wired and wireless communication protocols thanks to the general purpose input/output pins (for example, UART, I^2C, and SPI), low consumptions, requiring usually less than 10 W to run, which is even more important when it comes to battery-supplied devices as in this case, and a multi-core processor that allows concurrent tasks to be executed simultaneously.
The computer's OS is Raspberry Pi OS Lite, which is a GNU/Linux-based system that will be accessed through a Secure Shell (SSH) protocol.
The data acquisition software is divided into three layers:

- **Data Reader Layer:** communicates with the other sensor modules and reads the raw information from these.
- **Data Collector Layer:** Interpret the raw information and collect it creating a data structure.

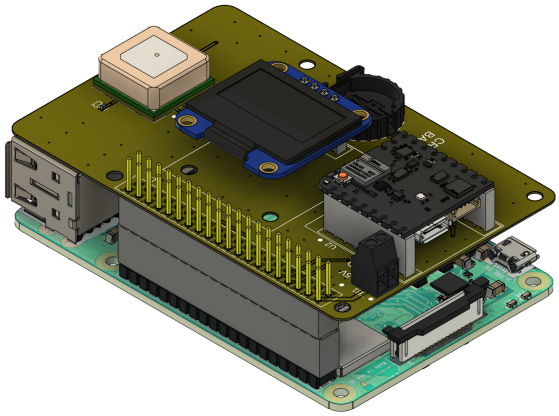


Figure 3.3: 3D representation of the DAM

- **Data Saver Layer:** Stores the data coming from the data collector layer in a permanent storage

This software is developed in Python language to speed up implementation time and be able to choose among many open-source libraries. Even though Python memory management is not really efficient, for demanding tasks the extensibility with C can be exploited.

Sensors

At this moment, the DAM is connected to four devices.

- **Nicla Sense ME:** Is a small and low-power board with industrial-grade sensors connected to the Raspberry Pi using the UART protocol. From this inertial measurements and environmental information are collected. The inertial data are provided by the BHI260AP sensor which has a six-axis IMU composed of a three-axis accelerometer and a three-axis gyroscope. A BMP390 is used as the pressure sensor, it is capable of performing in the range of 300 to 1250hPa. The environmental information is sensed by the BME688 sensor that measures temperature, humidity, and MOx air quality. Thanks to the UART protocol this board will guarantee a sample rate of 100 Hz for IMU measurements and 1 Hz for the environmental data.

- **SAM-M8Q GPS module:** Provides the reference in space (latitude, longitude, speed, altitude) and time to the DAM via I^2C protocol with a sampling rate of 4 Hz.

- **Raspberry Pi camera module:** Is connected to a USB port so that the numeric measurements are always paired with a reference image of the environment.

- **Motor Control Unit:** A Bluetooth Low Energy (BLE) connection allows the DAM to collect and store data from the motor control unit. From this device information on the battery, level of assistance, speed, and eventual motor errors

are transmitted to the data-gathering module. These measurements have a sampling rate of 1 Hz for motor errors and battery measurements and 3 Hz for the remaining ones.

- **OLED Display:** Permits the communication between the DAM and the rider, it's connected using I^2C.

All of these devices, except for the motor control unit, are contained by a 3D-printed shield that has been attached to the bike's handlebars.
Using a power analyzer, an average consumption of 3.5 W is recorded with short peaks of 5 W when the camera captures an image.

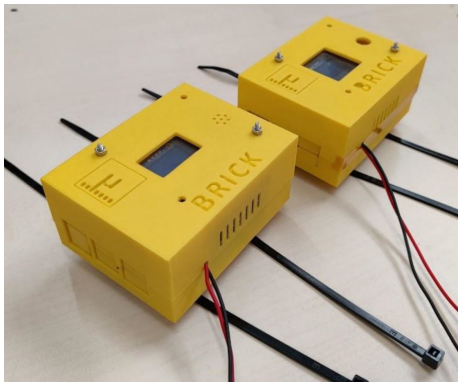


Figure 3.4: Sensing system housed in its 3D-printed shield

3.2 Experiments definition

To begin with, it is essential to determine both the number of terrains and the specific types of surfaces that will be characterized in this study. The selection process should consider terrains that are most representative of common urban environments to ensure practical applicability. This thesis will therefore concentrate on three of the most frequently encountered ground types typically found in urban settings: asphalt, gravel, and cobblestone. These surfaces were chosen due to their prevalence in city landscapes and their distinctive characteristics, which make them ideal for analysis and characterization.

Figure 3.5: Asphalt, gravel, and cobblestone terrains considered in this work

After the selection of terrains, it is necessary to identify and choose variables that may influence bike interaction with the ground. Recognizing these variables helps in gathering a comprehensive dataset that includes the most real-world variations possible, which is fundamental for the creation of a robust terrain recognition algorithm. The key parameters considered in this work are:

- **Terrain:** The experiments should be done on each of the selected terrains (asphalt, gravel, cobblestone).
- **Velocity:** Two different speeds have been chosen to characterize both a slow and fast gait ($15\frac{km}{h}$, $25\frac{km}{h}$).
- **Assistance level:** Three levels are chosen to be considered in the experiments (lvl. 0, lvl. 3, lvl. 5).
- **Damper:** This variable could make the measurements vary greatly, so data were collected in both states (locked, unlocked).

To correctly characterize the bike usage all the combinations of the identified variables have to be observed and used during the training phase. To be sure to have satisfying scenario coverage, table 3.1 has been studied ensuring that no critical scenario is overlooked.

Some of the initially planned experiments were not performed mainly for two reasons:

1. The unlocked shock absorber experiment has been done after some first analysis of data collected with the shock absorber locked. These preliminary analyses already suggested that the variation in the level of assistance doesn't affect measurements significantly, so it has been decided to only observe one level for the unlocked dumper experiments to speed up dataset collection.
2. During this work only a cobblestone-paved spot was available and it was not the perfect experimental environment. This spot is a short pedestrian area, making difficult a continuous collection of data. In fact, continuous reversal is needed and the presence of people made it challenging to conduct high-speed tests safely and consistently. As a result, the data collected in this environment was limited and, in some cases, less ideal for rigorous analysis.

Terrain	**Assistance [lvl]**	**Velocity** $[\frac{km}{h}]$	**Dumper**	**Timestamp**
Asphalt	0	15	Locked	05/07/2024 10:45
Asphalt	3	15	Locked	05/07/2024 10:30
Asphalt	5	15	Locked	05/07/2024 10:15
Asphalt	0	25	Locked	05/07/2024 14:10
Asphalt	3	25	Locked	05/07/2024 14:20
Asphalt	5	25	Locked	05/07/2024 14:30
Asphalt	0	15	Unlocked	*missing experiment*
Asphalt	3	15	Unlocked	10/09/2024 11:28
Asphalt	5	15	Unlocked	*missing experiment*
Asphalt	0	25	Unlocked	*missing experiment*
Asphalt	3	25	Unlocked	10/09/2024 11:39
Asphalt	5	25	Unlocked	*missing experiment*
Cobblestone	0	15	Locked	23/09/2024 10:28
Cobblestone	3	15	Locked	*missing experiment*
Cobblestone	5	15	Locked	23/09/2024 10:09
Cobblestone	0	25	Locked	*missing experiment*
Cobblestone	3	25	Locked	*missing experiment*
Cobblestone	5	25	Locked	23/09/2024 10:51
Cobblestone	0	15	Unlocked	23/09/2024 11:23
Cobblestone	3	15	Unlocked	*missing experiment*
Cobblestone	5	15	Unlocked	23/09/2024 11:09
Cobblestone	0	25	Unlocked	*missing experiment*
Cobblestone	3	25	Unlocked	*missing experiment*
Cobblestone	5	25	Unlocked	*missing experiment*
Gravel	0	15	Locked	05/07/2024 11:55
Gravel	3	15	Locked	05/07/2024 12:10
Gravel	5	15	Locked	05/07/2024 12:25
Gravel	0	25	Locked	05/07/2024 13:00
Gravel	3	25	Locked	05/07/2024 12:50
Gravel	5	25	Locked	05/07/2024 12:40
Gravel	0	15	Unlocked	10/09/2024 9:58
Gravel	3	15	Unlocked	*missing experiment*
Gravel	5	15	Unlocked	*missing experiment*
Gravel	0	25	Unlocked	10/9/2024 10:13
Gravel	3	25	Unlocked	*missing experiment*
Gravel	5	25	Unlocked	*missing experiment*

Table 3.1: Schedule and timestamp of the collected experiments

3.2.1 Experiments locations

The area selected for data acquisition is near the engineering campus of the University of Bergamo, in three spots where the ground is composed of one of the studied terrains (figure 3.6).

Figure 3.6: Experiments locations

From the GPS signal recorded by the e-bike is clear how shorter the cobblestone track is compared to the other two. This means that while for the two longest paths it's sufficient to go one way and back to cover around two kilometers, the other one must be traveled several times to reach the same distance.

3.3 Acquisition procedure

This chapter's section is dedicated to explaining the procedure to collect the dataset and will be useful for possible future new terrain introduction, or also for adding new training data for the currently considered terrain classes to improve their classification capabilities in not yet analyzed scenarios.

1. BRICK uses the e-bike battery, so is necessary to turn on the e-bike system. By doing this the data acquisition device is ready to be turned on.

2. To start collecting measurements the Raspberry Pi has to be switched on. To do so the headlamp button on the Polini controller has to be pressed and, as explained in section 3.1.1, it will switch on the computer, and so it will start data collection.

3. Once the DAM is turned on, the data acquisition software starts saving the sensed data in a CSV file named as the timestamp observed during the device powering up.

4. At this point is possible to perform the experiments and store the measurements in the BRICK. While collecting data is important to follow precise speed, level of assistance, shock absorbing setting, and terrain (as seen in table 3.1) so that they can best represent the chosen context and be significant during the analysis.

5. When the bike ride is over the headlamp button can be pushed again to stop the acquisition system.

6. Finally, in order to download the measurements, an external computer has to be connected to the Raspberry Pi using the Ethernet port and SSH protocol to explore the DAM file system. The data files are stored at this path in the single board computer: `home/microlab/control_unit/data_aggregation_module`. Here the CSVs, images, and logs are stored respectively in the `csv`, `img`, and `log` folders.

3.3.1 Notes on acquisition procedure

Startup time

At the beginning of a new experiment is important to consider that the DAM startup is not instantaneous. The Raspberry Pi, like any other computer, has a boot time in which the OS kernel is loaded and initialized, after that, the data acquisition software needs to be run introducing more power-on time. Another process that needs time is the GPS search for satellites which in some areas could require some time.
In this project experiments, a startup time of about a minute was waited when every data collection started to be sure to have clean measurements of all the considered variables.

GPS signal importance

The BRICK exploits the GPS signal to get the current timestamp. While this information is not necessarily relevant to data analysis, the system must find satellites upon startup. In fact, the measurements are stored within files named after the observed timestamp, so if this last one is not updated, files with a misleading name will be created, or even worse there could be a data overwriting problem.

Tire Pressure

A bike's parameter that has not been considered in this work is tire pressure. During the project, the inflation was at a constant value of 2 atm, but is plausible that by decreasing it, the tire has greater vibration dampening. To improve the scenario coverage of the algorithm, a specific study on tire pressure effects on the bike's

response to terrain interaction has to be done, and after that consider whether a model improvement is needed to adapt it to this variable.

Event reporting technique

Since the experiments are performed in an outdoor and sometimes unpredictable setting, it is useful to have a method to precisely track events in the dataset. The first tool used is a smartphone that can be used to annotate what is happening during the collection, but this is not precise in time and is dangerous to use during riding. The technique used for this purpose is to quickly switch between two assistance levels. This simple action creates in the dataset a clear and precise signal in time and is easy to perform during bike riding, then a description of the event can be annotated on the smartphone when is safe to use it.

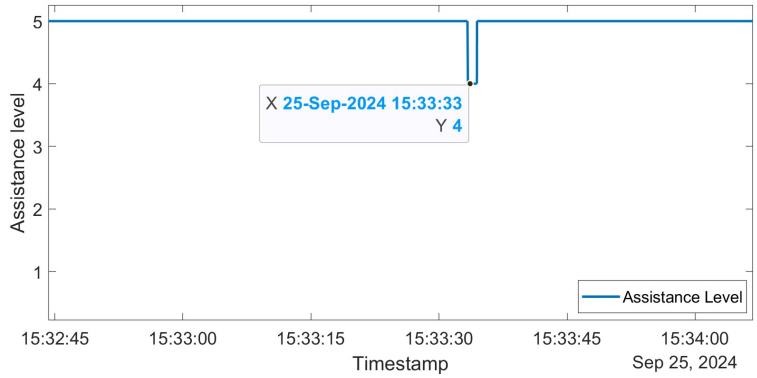


Figure 3.7: Event reporting example

Figure 3.7 shows how well this reporting method performs, allowing it to be accurate to the second.
Some examples of usage are to mark accurately the switch from one terrain to another or indicate when an interference with the experiment happened.

Chapter 4

Data Analysis and Preprocessing

The result of data acquisition is a dataset with a total of just under a million instantaneous measurements with a frequency of 100 Hz. These samples come from twenty-one experiments (eight on both asphalt and gravel and five on cobblestone), that have been used for the analysis and the training of the machine learning algorithm.
The collected data come in the form of CSV files with forty columns and a certain number of samples that represent the values observed by the sensor in an experiment of around five/ten minutes. The measurements stored in the forty columns are:

- **Timestamp:** timestamp in UNIX format of the acquisition;
- **Nicla_ts:** internal and relative timestamp of Nicla Sense ME, useful to check if the sensor is actually acquiring data every 10ms;
- **Nicla_accX, nicla_accY, nicla_accZ:** ADC reading of accelerometer on x,y,z axis;
- **Nicla_gyroX, nicla_gyroY, nicla_gyroZ:** ADC reading of gyroscope on x,y,z axis;
- **Nicla_temp:** temperature in Celsius degree;
- **Nicla_hum:** relative humidity in percentage;
- **Nicla_pres:** atmospheric pressure in Pascal;
- **Nicla_alt:** altitude estimated from atmospheric pressure in meters;
- **Nicla_iaq:** air quality index;
- **Nicla_gas:** MOx resistor for gas detection;
- **Nicla_roll, nicla_pitch:** roll and pitch angles estimated from accelerometer measurements (specified as not reliable by BRICK documentation);
- **Gps_time:** timestamp in UNIX format collected by GPS;
- **Gps_status:** boolean for the capability of localization based on the number of satellites found by GPS;

- **Gps_lon, gps_lat:** latitude and longitude in degrees;
- **Gps_alt:** altitude measured in meters;
- **Gps_speed:** speed measured by GPS in kilometers per hour;
- **Gps_numSV:** number of satellites found by the GPS;
- **Motor_soc:** bike's battery level in percentage;
- **Motor_distance:** meters traveled by bike since its first use;
- **Motor_res_range:** remaining distance before battery discharge in kilometers;
- **Motor_error_x:** eight boolean columns to flag eventual motor errors;
- **Motor_curr_cap:** remaining battery capacity in Ampere-hour;
- **Motor_curr_level:** current level of assistance;
- **Motor_spd:** bike's speed measured by the motor in 10 * kilometers per hour;
- **Motor_rider_pw:** power delivered by the rider in Watts;
- **Motor_assistance:** assistance delivered by motor in percentage;
- **Motor_cadence:** cadence of the motor in rpm;

The raw data collected needs to be cleaned and preprocessed to maintain in the dataset only useful information with the right transformations that allow and simplify the algorithm training.

4.1 Dataset Cleaning

4.1.1 Redundant columns

Some of the columns in the dataset represent the same feature, these columns have been analyzed to choose which to keep and which to remove from the data.

Altitude

This information is provided by the Nicla sensor, which estimates it using atmospheric pressure, and also by the GPS. The measurements sensed by the two devices have been plotted to understand which column to have in the dataset. To establish which is more precise, the altitude in a specific location (asphalt spot) has been checked and found to be around 190 meters. Looking at the altitude graph during this experiment, the GPS measurement seems more accurate and has been chosen.

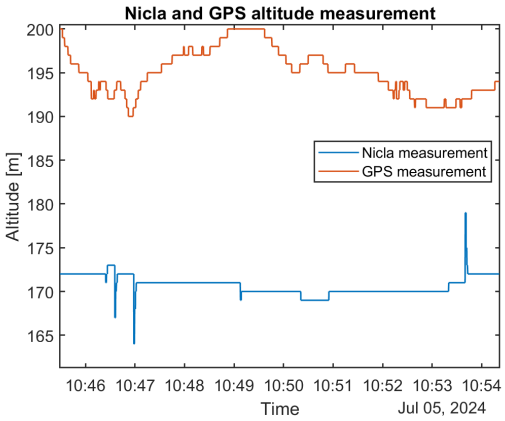


Figure 4.1: Comparison of altitude measurements on Nicla sensor and GPS

Speed

As for altitude, the bike's speed is represented by two dataset columns. The first measurements come from the GPS while the second from the motor sensor. Also in this case the choice has been made visualizing and comparing the measurements collected.

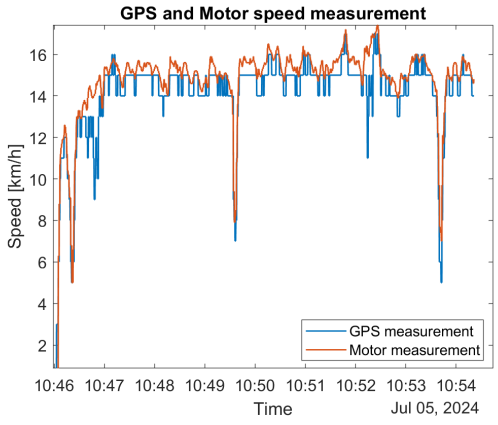


Figure 4.2: Comparison of speed measurements on GPS and Motor sensor

In this case, the two sensors roughly coincide in value, with the difference that GPS only uses integer values while the motor sensor provides a more defined output value. For this reason, the speed considered in the rest of the work will be the latter.

4.1.2 Motor errors

Eight of the forty columns in the dataset are flags for motor errors. In an offline setting as at this stage, an algorithm can be implemented so that it prints an error in the console if the experiments recorded a motor anomaly.

```
% for each of the eight error columns
for j = 0:7
    % Building the column name
    err_col_name = strcat('motor_err_',string(j));

    % Check for non-zero values
    if(any(preprocessed_data{i}.(err_col_name) ~= 0 & ...
            ~isnan(preprocessed_data{i}.(err_col_name))))
        % Creating and throwing an error message
        error_message = strcat('Motor error| error code:',string(j));
        error(error_message)
        continue;
    end
    % Removing the column from the dataset
    preprocessed_data{i}.(err_col_name)=[];
end
```

In an online application, these values will be checked iteratively to immediately report it to the user.

4.1.3 NaN handling

As already discussed in the data acquisition procedure explanation (section 3.3.1), the startup time of the sensing device is not instantaneous. Furthermore, also when the CSV file has been created, not all the sensor measurements are immediately available, so every recording will have at the beginning a certain number of Not a Number (NaN) values in some of the columns. The columns that are checked for missing values are the ones that have the main importance for the predictive algorithm, so timestamp, bike's speed, and the inertial measurements. The rows of the dataset that contain a NaN in one of these columns can be dropped since they will be samples captured at the initial startup.

4.2 Dataset Preprocessing

Once the data has been cleaned, some other transformations are needed to prepare data for the next analysis.

4.2.1 Measurements conversions

Some of the features use formats that are not clear for visualization and analysis, so these need to be converted.
First of all, the timestamps are stored in UNIX format. This representation is often used in computing and can be interpreted as the number of seconds elapsed since 00:00:00 UTC on 1st January 1970. Since this is not for sure the best way to visualize the timestamp, it has been converted to DateTime with the 'Europe/Rome' time zone.

Then the inertial features have been converted from ADC reading value to a more meaningful unit of measurement. To do so the datasheet of the Nicla sensor, and specifically the BHI260AP, has to be analyzed. The accelerometer can be set on

Sensitivity	S_{2g}	g_{FS2g}, T_A=25°C		16384		LSB/g
	S_{4g}	g_{FS4g}, T_A=25°C		8192		LSB/g
	S_{8g}	g_{FS8g}, T_A=25°C		4096		LSB/g
	S_{16g}	g_{FS16g}, T_A=25°C		2048		LSB/g

Figure 4.3: Accelerometer sensitivity parameters

four different scales, from a full-scale value of 2g to 16g. The sensor inside BRICK is set to 8g, so every accelerometer reading has to be divided by 4096 to transform the reading from an integer value to g.
The gyroscope measurements also need to be converted. The same procedure to

Sensitivity	R_{FS2000}	Ta=25°C, with default sensitivity correction in Fuser2 firmware applied		16.4		LSB/°/s
	R_{FS1000}			32.8		LSB/°/s
	R_{FS500}			65.6		LSB/°/s
	R_{FS250}			131.2		LSB/°/s
	R_{FS125}			262.4		LSB/°/s

Figure 4.4: Gyroscope sensitivity parameters

preprocess accelerometer columns is used for the gyroscope's ones, which uses a full-scale value of 2000°/s and so the measurements have to be divided by 16.4.
At the end of these transformations, the dataset has the inertial measurements respectively in g for the accelerometer and °/s for the gyroscope.

The last conversion performed during preprocessing involves speed measurement, this feature comes from raw data in $10 * \frac{km}{h}$. To have better interpretability during the analysis this column has been divided by 10 so that now speed is represented in $\frac{km}{h}$.

4.2.2 Norms computation

One way to get information about the intensity of accelerations and angular velocities independently of axes is to calculate the norm of these signals as:

$$\|a\| = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

$$\|\omega\| = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2}$$

Unluckily, the frequent reversals in the cobblestone experiments make unusable the

gyroscope norm. In fact, during the reversal, the handlebar is rotated and thus affects this latter measure significantly.
Regarding the accelerometer norm, this turns out to be a value that is independent of sensor position and orientation. It can be useful for terrain classification, so will be included as a feature in the dataset.

4.2.3 Reference frame rotation

Bike reference frame

The bike's reference frame, in this work, has been assigned according to the one used by the IMU.

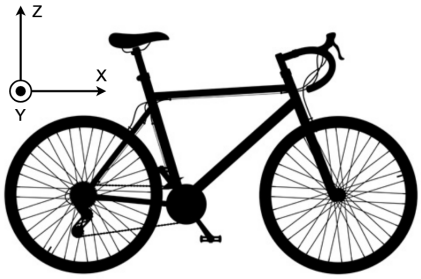


Figure 4.5: Bike with its reference frame

The X-axis corresponds to forward and backward accelerations and roll, the Z-axis represents vertical accelerations and yaw, and the Y-axis is related to lateral accelerations and pitch.

Sensor reference frame

Although, as mentioned earlier, the reference frame of the bike was chosen to agree with that of the sensor, these do not match precisely (figure 4.6).

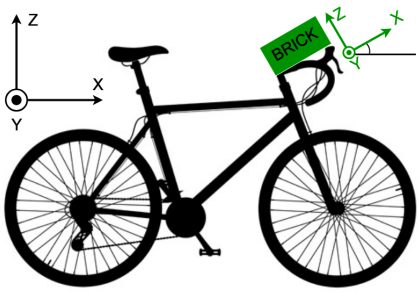


Figure 4.6: Comparison between bike (in black) and BRICK (in green) reference frames

The placement of the BRICK on the bike handlebars introduces an angle around the Y axis.
Due to this angle, the visualization and analysis could be not intuitive. Therefore it was decided to calculate the angle and to perform a rotation of the inertial data from the sensor to realign it with the bike frame.

Rotation using Rodrigues' formula

The Rodrigues' formula is an algorithm useful to rotate vectors in a three-dimensional space.
The formula states that given $\boldsymbol{v}$ a vector in $\mathbb{R}^3$ and a unit vector describing the rotation axis $\boldsymbol{k}$, the vector rotated of θ on $\boldsymbol{k}$ axis will be:

$$\boldsymbol{v_{rot}} = \boldsymbol{v}cos\theta + (\boldsymbol{k} \times \boldsymbol{v})sin\theta + \boldsymbol{k}(\boldsymbol{k} \cdot \boldsymbol{v})(1 - cos\theta).$$

Or using the matrix notation, as done in this work:

$$\boldsymbol{v_{rot}} = \boldsymbol{R}\boldsymbol{v}$$

where

$$\boldsymbol{R} = \boldsymbol{I} + (sin\theta)\boldsymbol{K} + (1 - cos\theta)\boldsymbol{K}^2.$$

In order to apply the appropriate rotation to the acceleration vector ($\boldsymbol{a}$) and angular velocity vector ($\boldsymbol{\omega}$) the rotation axis and the angle must be computed.
To realign the two frames, a small dataset containing samples of the bike in a static situation has been collected, so that the rotation matrix needed to transform the static acceleration measurement to the gravity vector ($\boldsymbol{g}$) can be calculated.

Since accelerations have g as the unit of measurement:

$$\boldsymbol{g} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

and the needed rotation matrix is $\boldsymbol{R}$ such that:

$$\boldsymbol{g} = \boldsymbol{R} \begin{bmatrix} a_{x_{static}} \\ a_{y_{static}} \\ a_{z_{static}} \end{bmatrix}.$$

The rotation axis can be computed by normalizing the cross-product between the static vector and the gravity vector:

$$\boldsymbol{k} = \frac{\boldsymbol{a_{static}} \times \boldsymbol{g}}{\|\boldsymbol{a_{static}} \times \boldsymbol{g}\|},$$

and then defining the skew-symmetric matrix useful for the matrix notation formula as:

$$\boldsymbol{K} = \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix}.$$

Since the Y-axis component is not changing in the rotation, the resulting axis will correspond to the Y-axis, with the opposite direction, so:

$$\boldsymbol{K} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

To find the rotation angle, the law of cosines applied to vectors can be used, thus:

$$cos\theta = \frac{\boldsymbol{a_{static}} \cdot \boldsymbol{g}}{\|\boldsymbol{a_{static}}\|\|\boldsymbol{g}\|}$$

$$\theta = cos^{-1}\left(\frac{\boldsymbol{a_{static}} \cdot \boldsymbol{g}}{\|\boldsymbol{a_{static}}\|\|\boldsymbol{g}\|}\right)$$

Finally, after having found both $\boldsymbol{K}$ and θ, the rotation matrix can be computed and used to transform the accelerometer and gyroscope values:

$$\boldsymbol{a_{rot}} = \boldsymbol{R}\boldsymbol{a}, \qquad \boldsymbol{\omega_{rot}} = \boldsymbol{R}\boldsymbol{\omega}$$

In the end, the Z-axis component of acceleration can be centered to 0 by removing the gravity component, to have a better data visualization:

$$a_{z_{centered}} = a_z + 1.$$

4.2.4 Inertial measurements normalization

In this work, it has been decided to implement the terrain-recognition algorithm using a single classifier. To achieve this, it is of main importance to make the signals vary only as the terrain changes, and not with the other external parameters. One of these efforts is to try to decouple the inertial measurements from the speed at which the bike is going. Looking at the IMU signals in the different experiments is clear that a faster pace leads to wider signals.

To keep the signal amplitude independent of speed, it has been decided to normalize all inertial measurements by the instantaneous speed experienced when these have been collected.

$$a_{x_{norm}} = \frac{a_x}{v}, \qquad a_{y_{norm}} = \frac{a_y}{v}, \qquad a_{z_{norm}} = \frac{a_z}{v}$$
$$\omega_{x_{norm}} = \frac{\omega_x}{v}, \qquad \omega_{y_{norm}} = \frac{\omega_y}{v}, \qquad \omega_{z_{norm}} = \frac{\omega_z}{v}$$

4.3 Data Visualization

With all the data preprocessed and ready to be used, a custom function for visualizing the whole dataset has been created. This function takes as input the dataset containing all the experiments, and for every terrain displays four plots regarding different types of data.

4.3.1 IMU data

The features included in this category are seven: the six-axis IMU and the norm of acceleration.

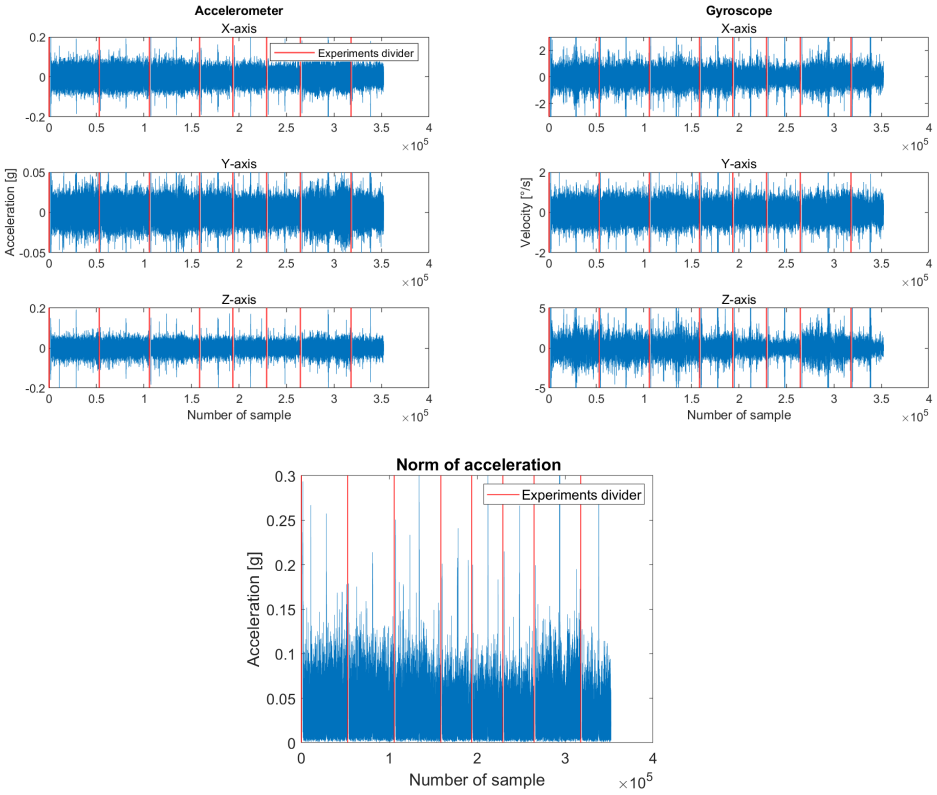


Figure 4.7: Examples of IMU signals on gravel

It can be noticed that the inertial signals have high-frequency components. To make this data meaningful and utilize it to classify the terrains it will be necessary to group the measurements and compute significant values for the algorithm. Figure 4.7 shows an example of the displayed plot, in this case for gravel terrain. Although graphs of the raw inertial data are difficult to interpret, comparing these for the three soils already reveals some differences confirming that inertial measurements can be a good input for classification.

4.3.2 Environmental data

Another subset of features in the dataset is the environmental ones (figure 4.8).

These measurements won't be used in this specific project, anyway are useful to track the atmospheric conditions during the ride and could be useful in future works.

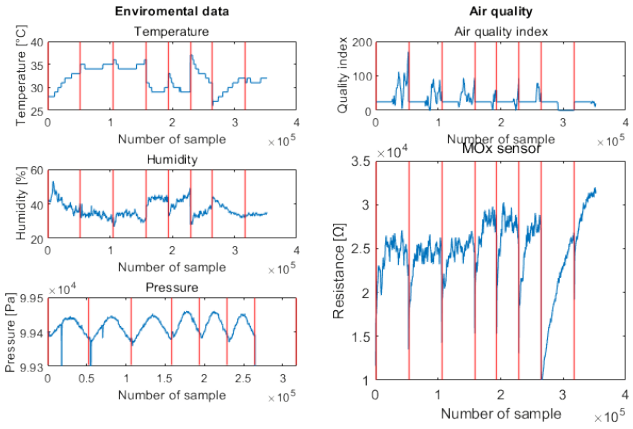


Figure 4.8: Examples of environmental signals on gravel

4.3.3 GPS data

The data from the GPS (figure 4.9) will not be needed by the recognition algorithm, however, it is very useful during analysis to have context about where the experiments are coming from.

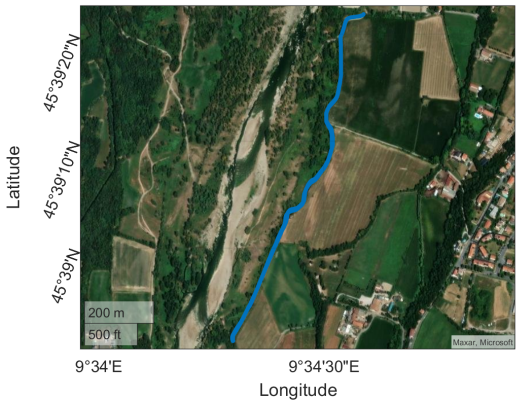


Figure 4.9: Example of GPS signal on gravel

4.3.4 Motor data

Finally, the last group of features is the one with measurements relative to the motor and the battery (figure 4.10).

Since there are no significant differences in these signals among the three terrains, the only feature useful for the algorithm will be the bike's velocity.

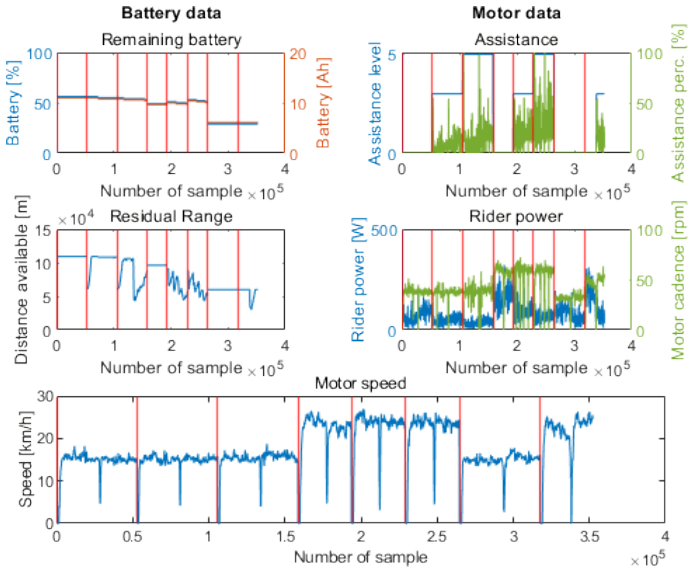


Figure 4.10: Example of motor and battery signals on gravel

4.4 Feature extraction

During the feature extraction phase, the dataset dimensionality is reduced in order to keep only the columns that contain significant information for the classification algorithm. The visual inspection of data explained in section 4.3 allows us to discard the features that don't regard inertial measurements, which is already a great cut to the dataset dimension.

4.4.1 Useful inertial measurements for terrain recognition

Of the seven inertial features kept, not all are of interest in the area of terrain recognition.
An analysis of which device and which axis is useful to observe during this task has been performed by DuPont [15]. In his paper, an autonomous ground vehicle with four wheels is studied, and it is stated that: "The vibration due to the vertical wheel displacements is completely characterized by the vertical, roll, and pitch motions and here are considered to be the vertical acceleration $a_z(t)$, the pitch rate $\omega_y(t)$, and the roll rate $\omega_x(t)$". This analysis can be extended to the case of a bike, with some changes.
First, the roll rate is not useful in a two-wheel vehicle. In fact, having just two wheels, the only rate influenced by the terrain is the pitch that changes when the front and back tires are not at the same height, for example, on rough roads as gravel.
The acceleration on the vertical axis well describes the vertical displacement, but, because the handlebar on which the sensor is fixed is diagonally connected to the front wheel, some of the information regarding vertical displacement is also distributed to the x-axis of the accelerometer.

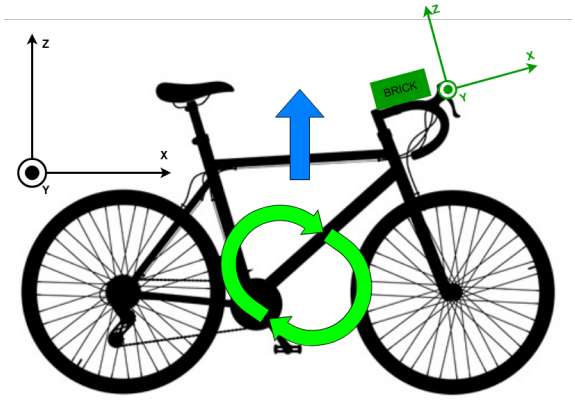


Figure 4.11: Vibrations that characterize the ground beneath, pitch rate (in green), and vertical displacement (in blue) on bike (in black), and BRICK (in dark green) reference frames.

The conclusions reached from this analysis also correspond by looking at the collected data. The features that change most visibly as the terrain changes are the y-axis of the gyroscope and the x-axis and z-axis of the accelerometer.

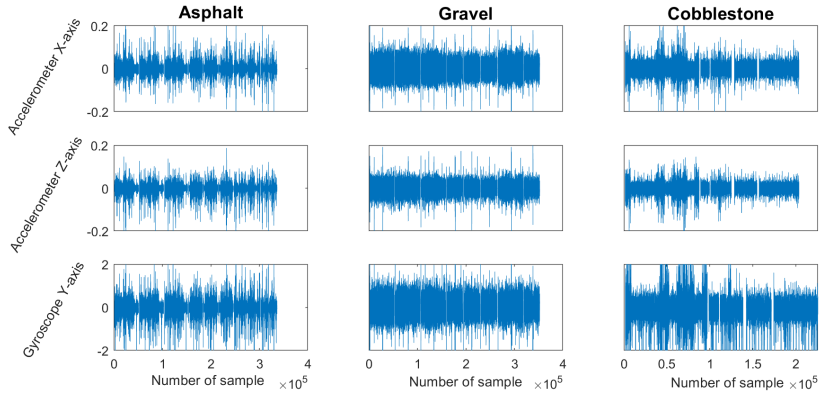


Figure 4.12: Inertial measurements coming from different terrains comparison

Comparing the inertial signals coming from different terrains in figure 4.12, it can be seen that on average gravel has the most intensity, followed by cobblestone, and finally asphalt signals have low energy with some peaks caused by unevenness of the ground. After this column selection process, the dataset has been reduced from forty variables to just four, which are the three chosen previously and the norm of acceleration.

4.4.2 Temporal feature extraction

The inertial signals as collected during the experiments are not that significant for a classification task. In order to have meaningful data that can be fed to the machine-learning algorithm, all the considered IMU signals has been grouped into segments. After this, for each segment values capable of summarizing the contained information are computed.

Segmentation algorithm

In the literature studied during the preliminary phases of this project, the most used technique for signal fragmentation is to divide the data into segments of a constant temporal window, usually around one second.
In this work, a new methodology has been ideated. The change of approach has been done because of the wider range of velocities experienced compared to the analyzed papers. While the autonomous vehicles considered in literature usually keep a constant speed, a normal bike ride can have a relatively high change of pace. To build a robust algorithm that is independent of the speed, instead of dividing the signal into segments of constant length, a segmentation based on a constant number of wheel revolutions has been implemented. This approach will ensure that each segment will have information regarding a specific space movement, instead of a time frame.

```
function [outData] = create_rev_groups(inData,R_WHEEL,N_REV_GROUP)
% inData has three tables (one for every terrain)
for i=1:length(inData)
    start_idx = 1;
    group = 1;

    while start_idx<=height(inData{i})
        % Computing rev/s at current velocity
        rev_s = (inData{i}(start_idx,:).motor_spd/3.6)* ...
            (1/(2*pi*R_WHEEL)); % /3.6 because speed is in km/h

        % Computing time to complete N_REV_GROUP revolutions and
        % group dimension
        group_dim = floor((N_REV_GROUP/rev_s) * 100);
        % multiplied by 100 because a sample represents 10ms

        end_idx = start_idx + group_dim; % group end index

        % Assigning group value to samples from start_idx to end_idx
        % of outData{i}
        % .
        % .
        % .

```

```
25          group = group+1;
26          start_idx = end_idx+1;
27      end
28  end
29  end
```

The function, in addition to the input dataset, also is fed with the radius of the wheel and the number of revolutions that every group contains, so that the function can be used also with other bikes and the group dimensions can be tuned.

Computing temporal statistics

A way to characterize signal segments is to compute statistics on these in the time domain. Looking at the examined research it has been chosen to compute for every one of the four signals:

- Mean
- Variance
- Maximum value
- Minimum value
- For four different thresholds, the number of times it is crossed.

The thresholds have been set manually by visual inspection, choosing values that are frequently crossed just by one of the terrains for each variable.
Considering these calculations for the four kept signals, the dataset of grouped samples has for every row a total of 32 features (eight values for each inertial measurement).

4.4.3 Frequency analysis

Some other information regarding the characterization of the terrains interactions with the bike can be found by looking at the signals frequency response.
Having more experiments with different scenarios for each terrain is necessary to understand if varying these parameters during the data collection, the frequency response change. For this reason during frequency analysis, the different experiments will be analyzed separately. In addition to this, every experiment signal will be also examined on its constant speed portion, removing the parts where the ride pace is too different from the one expected, such as at the beginning or during events that interfere with the experiment causing a slowdown. Apart from this the rest of the preprocessing procedure remains the same described above.

Windowed Fast Fourier Transform

The method chosen for frequency analysis is the Fast Fourier Transform (FFT), which is an algorithm to compute the Discrete Fourier Transform (DFT). The result of this computation will be the intensity distribution of the signal in the frequency domain.
Applying the FFT directly to the whole signal in this case is problematic because of the length of the signal and the low range of frequency observable.
The sensor used in the project has a sampling rate of 100 Hz and for the Nyquist theorem the maximum band limit is:

$$f_{max} = \frac{f_{sampling}}{2}$$

This means that all the input data of the FFT is used to compute the frequency response in the first 50 Hz, causing a response too defined that is hard to analyze.
To solve this situation the following algorithm has been implemented.

```
function [ffts_mean,f] = windowed_fft(X,window_length, overlap)
    N = length(X);
    Fs = 100; % Sampling frequency

    hann_window = hann(window_length); %hanning the window
    num_segments = floor((N - overlap) / (window_length - overlap));

    fft_segments = [];
    for j = 1:num_segments
        % Computing segment interval
        start_idx = (j - 1) * (window_length - overlap) + 1;
        end_idx = start_idx + window_length - 1;
        if end_idx > N
            break;
        end

        % Extracting the segment and windowing
        segment = X(start_idx:end_idx);
        windowed_segment = segment .* hann_window;

        % Computing segment FFT
        fft_segment = fft(windowed_segment);
        fft_segment = fft_segment(1:floor(window_length/2))';
        % Adding segment FFT to the list
        fft_segments = [fft_segments; abs(fft_segment)];
    end

    ffts_mean = mean(fft_segments); % means of ffts
    f = (0:floor(window_length/2)-1) * Fs / window_length;
end
```

The function divides the signal into smaller segments and computes the FFT on these. A Hann window is applied so that the signal cutting does not introduce spectral leakage giving less importance to measurements on the edge of the segment. Finally, the mean of the list of FFT is returned as a result. In this way, the output will still be computed on the whole signal, but it will have a lower definition.

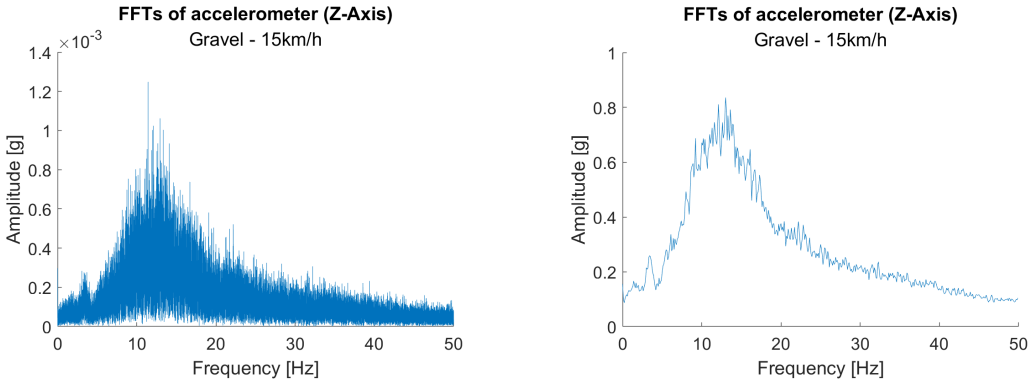


Figure 4.13: Example of FFT on the whole signal (on the left), and with windowing (on the right)

In figure 4.13 it can be seen how easier is to understand frequency response using this technique.

Frequency response difference with changing level of assistance

The first parameter that could influence frequency response is the level of pedal assist selected during the experiment. The experiments with the same terrain and the same speed have been compared to see if this factor could introduce some differences.

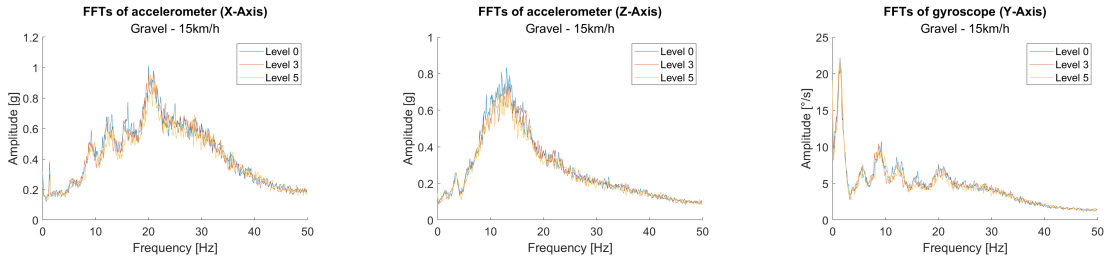


Figure 4.14: Comparison of frequency response with different levels of assistance on three gravel signals at 15km/h

The comparison shown in the image above has been performed for every terrain and speed combination, and no one of these highlighted differences as the selected level changed. Therefore, it was determined that this parameter does not affect the collected data.

Frequency response difference with changing riding speed

The same procedure has been done with changing speed.

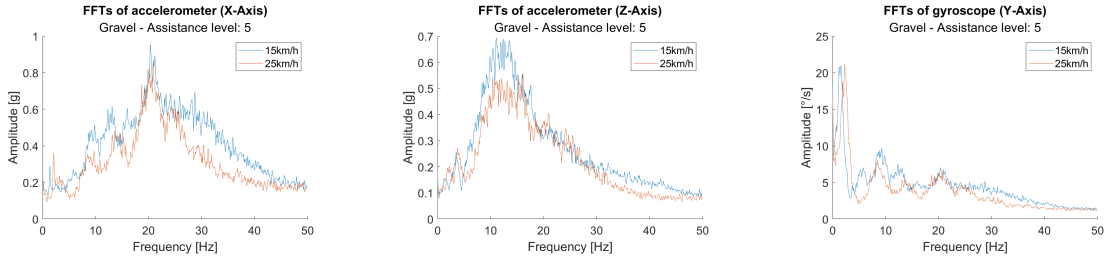


Figure 4.15: Comparison of frequency response with different speeds on three gravel signals

While the level of assistance doesn't affect the frequency response, it seems that speed does. These differences will be taken into account during the frequency feature extraction phase. However, despite being different, it is good to notice that the normalization of measurements by velocity allowed for comparable amplitudes during frequency analysis.

Frequency response difference with locked and unlocked damper

Finally, the last parameter that could introduce variations in the frequency response of the three signals is the status of the damper.

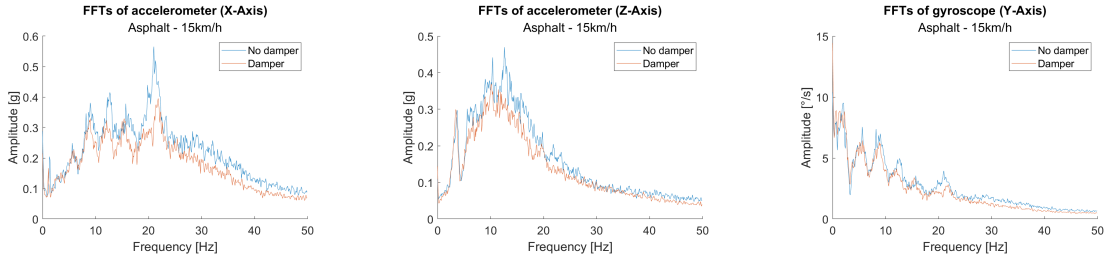


Figure 4.16: Comparison of frequency response with damper locked and unlocked on three asphalt signals

In this analysis the signals shown in image 4.16 come from asphalt instead of gravel as the other examples because is in this terrain that is easier to see the effect of damper. Looking at the graphs it can be seen that the damper reduces the amplitude of the frequency response and so, like for speed, needs to be taken into account during the feature extraction.

4.4.4 Frequency feature extraction

The method adopted to extract significant features from the frequency response of the inertial measurements is to identify through visual inspection the frequency bands in which the result of the FFT is different enough among the different terrain samples. Once these ranges are found, a significant feature could be the integral of

the frequency response in that specific band.

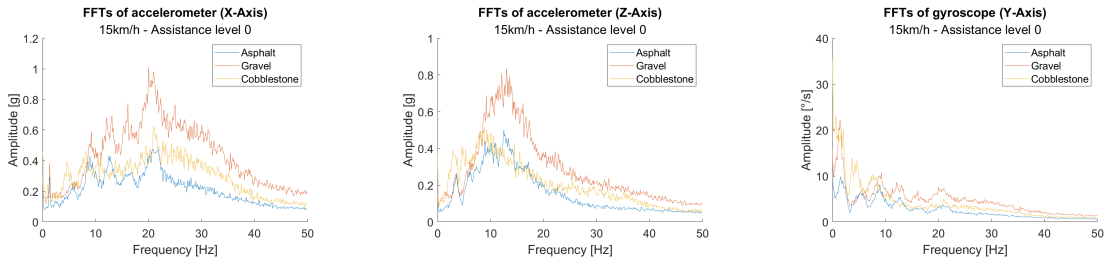


Figure 4.17: Comparison of the frequency response of different terrains on the three analyzed sensor signals

Analyzing the results of the signals FFT, some clear differences are visible as the ground beneath changes. This is true when the same ride condition is maintained, but, as seen above, some parameters affect the bike vibration while maintaining the same underlying terrain such as the dumper status and even more the speed kept during the ride. Since in this work, it has been decided to train a single model that will be valid in any condition, during the feature extraction process the sensor signal frequency response of a terrain will be represented by a range of possible values, instead of a single line.

Frequency features on the X-axis of the accelerometer

The graph in figure 4.18 shows the intervals in which every experiment spectrum of the accelerometer X-axis is contained for the three terrains.

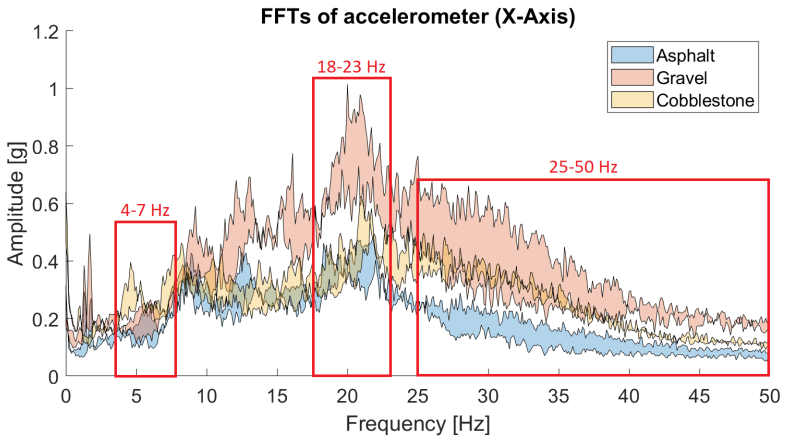


Figure 4.18: Frequency response intervals on X-axis of the accelerometer

For this signal, three bands have been chosen as significant enough to characterize the three kinds of ground.

- **4-7 Hz:** In this range the cobblestone response is more intense than the other two.

- **18-23 Hz:** The three frequency responses are distinct in this interval.
- **25-50 Hz:** The high-frequency noise, although it does not represent a definite characteristic of the terrains, can divide the response of the asphalt very well from that of the gravel and cobblestone.

Frequency features on the Z-axis of the accelerometer

In the same way explained above, the signals coming from the z-axis of the accelerometer have been analyzed (figure 4.19).

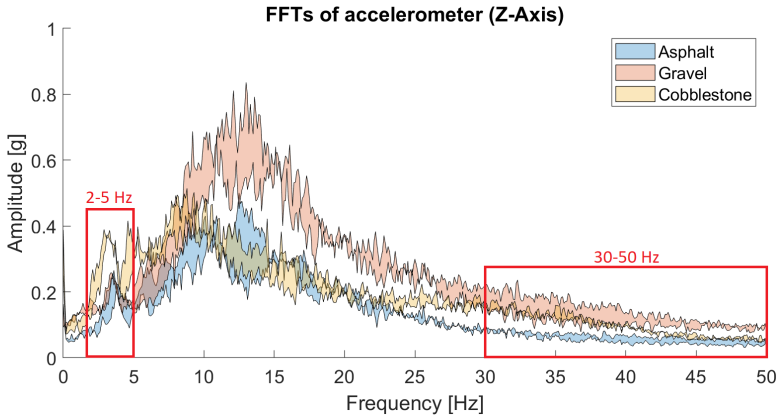


Figure 4.19: Frequency response intervals on Z-axis of the accelerometer

In this case, two intervals have been chosen for the characterization of the three classes.

- **2-5 Hz:** Similarly to the accelerometer signal on the x-axis, in this low-frequency band the cobblestone has a different footprint than the other two signals.
- **30-50 Hz:** Also on this axis of the accelerometer the high-frequency noise can distinguish asphalt and, though less clearly, also gravel and cobblestone.

It has been chosen not to use the range around 15 Hz because it only identifies gravel, which is already accomplished by other features, and in addition, could make mistakes for cobblestone and asphalt.

Frequency features on the Y-axis of the gyroscope

Finally, also the gyroscope signal used during the classification has been analyzed in order to discover useful features for the model.

In these signals, two intervals have been found to be interesting for feature extraction.

- **1-3 Hz:** This range at very low frequency proved capable of distinguishing the asphalt from the other two terrains.

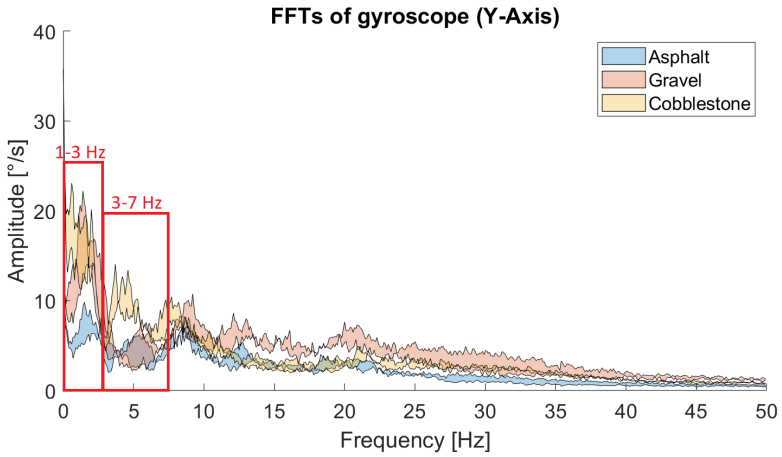


Figure 4.20: Frequency response intervals on Y-axis of the gyroscope

- **3-7 Hz:** Immediately in the frequencies next to the first interval, signals from the cobblestone ground leave an imprint that differentiates from the other classes.

4.4.5 Processed dataset

At the end of the feature extraction process, the raw dataset described at the beginning of chapter 4, has been transformed to be fed as input to the classification algorithm.
The modifications applied are:

1. The columns that do not refer to inertial measurements won't be used during the classification so have been dropped. Furthermore, even among the inertial measures, only those capable of describing the interaction between the bike and the ground were kept, which are: the accelerometer x-axis and z-axis, the gyroscope y-axis, and the acceleration norm.

2. Every row of the dataset is no longer an instantaneous measurement, but a gathering of a certain amount of sensed values with dimension computed by the algorithm presented in section 4.4.2. The choice made during this work is to have groups of three-wheel revolutions since it is a good trade-off between prediction responsiveness and the amount of data per sample.

3. The columns of the dataset have switched from representing the value collected by a specific sensor to a temporal or frequency feature, calculated over a group of values.

   (a) **Temporal features:** for every signal kept in the dataset these values are computed:

   - Mean
   - Variance
   - Maximum value

- Minimum value
- For four different thresholds, the number of times it is crossed.

So a total of 32 temporal features.

(b) **Frequency features:** The dimension of the groups created by the gathering algorithm usually is between one hundred and two hundred. To keep the output of the FFT independent from the input dimension, it has been decided to compute the 200-point FFT.
For the three computed spectra, the values kept as features are:

- **Y-axis gyroscope:** $\int_1^3 A_{gyro_Y}(f)\,df$, $\int_3^7 A_{gyro_Y}(f)\,df$
- **X-axis accelerometer:** $\int_4^7 A_{acc_X}(f)\,df$, $\int_{18}^{23} A_{acc_X}(f)\,df$, $\int_{24}^{50} A_{acc_X}(f)\,df$
- **Z-axis accelerometer:** $\int_2^5 A_{acc_Z}(f)\,df$, $\int_{30}^{50} A_{acc_Z}(f)\,df$

So a total of 7 frequency features.

Chapter 5

Terrain Recognition Algorithm

5.1 Preliminaries to the classification algorithm

5.1.1 Learning method selection and multi-class extension

The method used to classify has been chosen among the ones proposed by the studied literature. The final choice came down to a multi-class SVM (Support Vector Machine) because of its simplicity and ease of interpretation. This algorithm is capable of finding an optimal hyper-plane that maximizes the margin between the closest points with different classes. This classification method will be investigated further in Appendix A.1.
The SVM is a binary classifier, so its task is to divide two classes. To expand this functionality to a multi-class case, as in this project, an ensemble of more binary SVM classifiers is needed. One of the most widely used strategies for dealing with multi-class problems is error-correcting output coding (ECOC) [16]. This technique is composed of three phases: encoding, binary classifier learning, and decoding.

Encoding

The encoding phase consists of the construction of the code matrix. This matrix will be responsible for describing how the classes and the binary learners are linked. In this matrix, each row represents a class, and each column represents a binary classifier. The valid values in the matrix are 1 which indicates a class of the binary learner, -1 which refers to the other class, and 0 when the class is ignored by the learner. Some of the possible coding designs are:

- **One-versus-all** in which every binary learner is trained to distinguish between a class and all the others;
- **One-versus-one** that have a binary learner for every pair of classes, ignoring the other ones;
- **Binary-complete** partitions all the classes into all binary combinations without ignoring any class;
- **Ternary-complete** creates all the ternary combinations, including also the possibility of ignoring a class;

The encoding design chosen is the one that ensures the wider hamming distance between the rows, that is the ternary-complete. This method could be too verbose in the case of several classes, but since there are only three of these, the solution is feasible.

	L1	**L2**	**L3**	**L4**	**L5**	**L6**
Asphalt	-1	-1	0	1	-1	-1
Gravel	1	-1	-1	-1	0	1
Cobblestone	0	1	1	1	1	1

Table 5.1: Encoding matrix of ternary-complete design.

The ternary-complete encoding requires six binary learners:

1. Gravel vs Asphalt;
2. Cobblestone vs Asphalt∨Gravel;
3. Cobblestone vs Gravel;
4. Gravel vs Cobblestone∨Asphalt;
5. Cobblestone vs Asphalt;
6. Asphalt vs Cobblestone∨Gravel;

Binary learners training

Once the six binary learners needed have been defined, an SVM for each one is trained by feeding them with the dataset transformed as described in table 5.1 in order to have just two classes.
The training of the SVMs will be further explained later in section 5.2.1.

Decoding

The decoding phase starts with the creation of the prediction vector composed of the results of the six binary learners. Once this vector has been constructed, it is compared to the valid codewords written in the encoding matrix. The resulting class will be the one with the lower Hamming distance to the prediction vector.

5.1.2 Train and Validation split

During machine learning model training one of the problems that you may encounter is overfitting, that is the fitting of a model too specific for the training data, and so not capable of generalizing to a real-world situation. A standard procedure to avoid this issue is the train-validation-test split of the dataset.
By dividing the starting dataset into these subsets is possible to use the:

- **Train set:** to learn and optimize the variables that allow a good predictive model;

- **Validation set:** to make assumptions on the model performance and tuning the model parameters;
- **Test set:** to evaluate the performance of the final tuned algorithm on data that it has never seen.

The dataset considered until now composed of twenty-one experiments on different terrains will be split just into train and validation subsets, while the test data has been collected separately as described in the online classification section (5.3.1).
The train-validation split follows a 70%-30% division which guarantees both enough data for the model training and a sufficient subset for the performance evaluations during the validation phase.
This split can not be done just by taking the last 30% of the dataset for the validation, because it wouldn't be representative of all the experiment conditions. What has been done instead is to take that percentage of data from each singular experiment, so that the validation subset will be comprehensive of all the possible riding parameters changes.

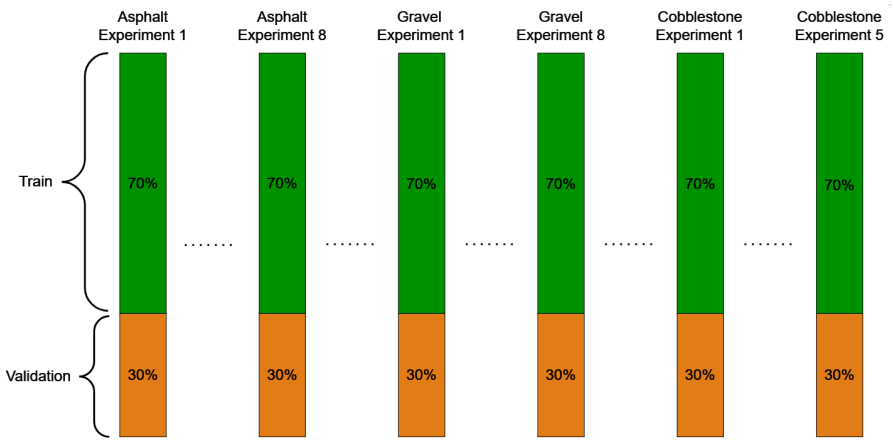


Figure 5.1: Train-validation split technique

5.1.3 Feature selection

In section 4.4.5 is shown how after cleaning, preprocessing, and extracting values, the dataset ends up having 32 temporal features and 7 in the frequency domain. Is clear how the feature extraction phase has made the dataset dimensionality explode despite the previous steps already reducing the dataset to just a few sensor values. For this reason, it has been investigated to understand if all of these features were actually important for the algorithm task. This process has been done by fitting a regularized model on a multi-class SVM.

Lasso Regularization

Regularization is a method that adds penalization to the residual sum of squares controlled by a hyper-parameter (λ). The penalization tends to bring the coefficient

near zero, excluding the ones that are not significant enough for the classification.

$$\text{RSS} = \sum (y_i - \hat{y}_i)^2$$
$$\min \quad J(\boldsymbol{\beta}) = \text{RSS} + \lambda \sum |\beta_i|$$

When it comes to feature selection, Lasso regularization is preferred because it penalizes the absolute value of the coefficients instead of the square like the Ridge one. This choice makes it easier to bring a coefficient exactly to 0 instead of having small values.

The amount of regularization is decided by setting the value of λ. When $\lambda = 0$ no regularization is introduced, while the more this hyper-parameter is increased the more penalty is introduced.
Is clear that the choice of λ is critical, and so it has been done by validation. More values for the parameter have been tried and each performance on the validation subset has been recorded, this permits the choice of a value that introduces a penalization, but at the same time keeps the classification capability.

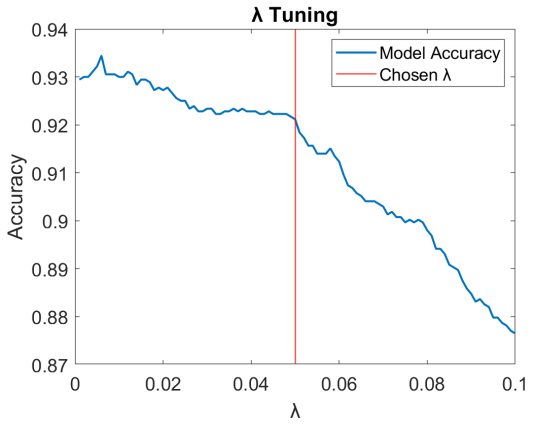


Figure 5.2: Tuning of the λ hyper-parameter

The value chosen by visual inspection is $\lambda = 0.05$ and it only reduces the performance of this algorithm on the validation dataset to less than 1% with respect to the model that uses all the features available.
Since this is a multi-class algorithm, the regularization is applied singularly to each of the six binary learners. It has been decided to consider meaningful every feature whose coefficient is different from 0 at least in one of the classifiers. This procedure drops the number of columns that will be used to nineteen, which represents a halving from the previous situation.
This selection technique is likely too lax as well, and additional features could be removed if necessary, but since the cut has already been quite extensive it has been decided to keep it as described.
The features that will be used in the final classification algorithm are:

- **mean_nicla_gyroY:** mean of the values registered on the Y-axis of the gyroscope per sample;

- **var_nicla_accX:** variance of the values registered on the X-axis of the accelerometer per sample;
- **var_norm_acc:** variance of the values registered on the norm of accelerometer per sample;
- **max_nicla_accZ:** maximum value registered on the Z-axis of the accelerometer per sample;
- **min_nicla_accZ:** minimum value registered on the Z-axis of the accelerometer per sample;
- **th1/th2/th3/th4_cross_norm_acc:** Number of threshold crossing in the sample for all four on the norm of the accelerometer;
- **th2/th3_cross_nicla_accX:** Number of threshold crossing in the sample for the second and the third one on the X-axis of the accelerometer;
- **th3/th4_cross_nicla_gyroY:** Number of threshold crossing in the sample for the third and the fourth one on the Y-axis of the gyroscope;
- **GyroY1-3Hz:** Integral of the frequency response of the sample between 1 and 3 Hz for the Y-axis of the gyroscope;
- **GyroY3-7Hz:** Integral of the frequency response of the sample between 3 and 7 Hz for the Y-axis of the gyroscope;
- **AccX4-7Hz:** Integral of the frequency response of the sample between 4 and 7 Hz for the X-axis of the accelerometer;
- **AccX24-50Hz:** Integral of the frequency response of the sample between 24 and 50 Hz for the X-axis of the accelerometer;
- **AccZ2-5Hz:** Integral of the frequency response of the sample between 2 and 5 Hz for the Z-axis of the accelerometer;
- **AccZ30-50Hz:** Integral of the frequency response of the sample between 30 and 50 Hz for the Z-axis of the accelerometer;

5.2 Offline classification problem

5.2.1 SVM training and tuning

Hyper-parameters tuning

In this section, the hyper-parameter tuning procedure applied to the binary learners will be explained. Further explanation on the SVMs will be done in A.1. An SVM has some parameters that can change its behavior during the classification:

- **Standardization:** A binary value to decide if standardize the input matrix. This technique is usually suggested so that all the features are on the same scale. Since this model is based on distances, the standardization guarantees that all the features will be considered equally. Furthermore, the training time is also highly reduced on standardized data.
- **Kernel Function:** The SVM model can use different kernel functions to divide the classes. This functionality could be useful if the relationship between the features and the classes is nonlinear.
- **Kernel Scale:** This hyper-parameter, often called γ, controls the scaling of the kernel if a different from the linear one has been chosen.
- **Box Constraint:** Usually represented as C, allows the control on the trade-off between margin width and misclassification. From a visual perspective, this parameter controls how wide the margin of the classifier can be, so a large value of C will make the model search for a hyper-plane with small margins, while small values will prefer generalization permitting wider margins on the classificator.
- **Outlier Fraction:** This parameter can be set to decide the expected fraction of outlier in the dataset. The model will fit the model for the first time, after that it will find the selected percentage of samples with a larger magnitude in the gradients. By doing this the outliers are found, so the model is retrained excluding these values.

All of these hyper-parameter has to be set to optimize the performance of every binary learner.
As the first parameter, standardization has been applied since it has many positive sides to the classification performance, as well as facilitating model interpretability during feature importance calculation.
Some different kernel functions have been tested, even though during data visualization the relationship between the various features and the classes seems linear. The tests done with the validation technique show that not using a kernel trick leads to better performances. This choice will also guarantee a better model interpretability. Since the kernel trick has not been used, the γ parameter does not need to be set.
It has been decided to set an outlier fraction since the dataset is large enough. This parameter has been chosen to be 5% which is a standard value when choosing an outlier fraction.
In the end, the Box Constraint value has been chosen with cross-validation. The parameter values search is done mainly with three techniques:

- **Grid search:** Some specific values values are given and the algorithm tests every possible value, and every possible combination in the case of more parameters to tune, and after looking at all the performances finds the best one.
- **Random search:** Given an interval of feasible values, the algorithm computes the model's performance by searching randomly in the specified range for a fixed amount of times. A graph of the computed performance can be plotted to have an idea of how it changes as the parameter is varied.

- **Bayesian search:** In contrast to the other two techniques, Bayesian search memorizes the previous performance and fits an estimated function. In this way, the algorithm has an idea of the specific range in which the hyper-parameter can find a minimum.

To efficiently find a value for the Box-Constraint parameter, a Bayesian search has been done.

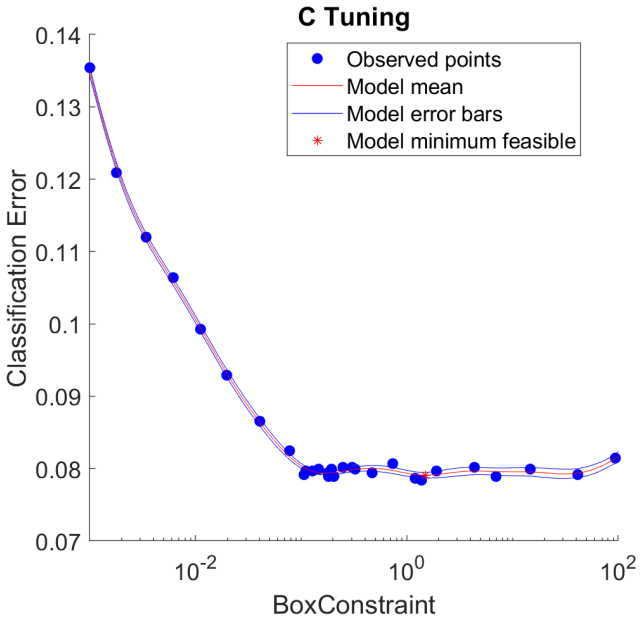


Figure 5.3: Bayesian optimization on C hyper-parameter

The graph in figure 5.3 shows that an optimal value is already found in the interval between 0.1 and 1, and then the performances are nearly constant with the increasing of C. A low value is preferred because it has more generalization capability. With this last tuning process that allowed to assign to C a value of 1, all the SVM parameters have been chosen and so are used for the binary learners of the multi-class model.

Regarding the ECOC multi-class model, as already explained in section 5.1.1, a ternary-complete encoding has been used. Another modification with respect to the standard model has been to change the prior probability. By default, the multi-class model considers a prior probability during the classification task which is based on the distribution of the classes in the training data. This functionality needs to be removed in this case because the cobblestone terrain is under-sampled compared to the other two classes, so it could be more negatively affected by this model property. Thinking about the real-world problem, all the terrain should have the same prior probability, so it has been changed to have a uniform distribution.
Once the class boundaries have been defined by the training procedure, the ECOC model allows a posterior probability function to be fitted. The output of this function will be a matrix that for every sample gives the percentage probabilities of

belonging to each class. These values represent an indication of the confidence of the prediction and will be useful during the online classification algorithm.

Model performance on validation set

The previous choice and tuning processes can be evaluated by looking at the performance of the model on the validation set.

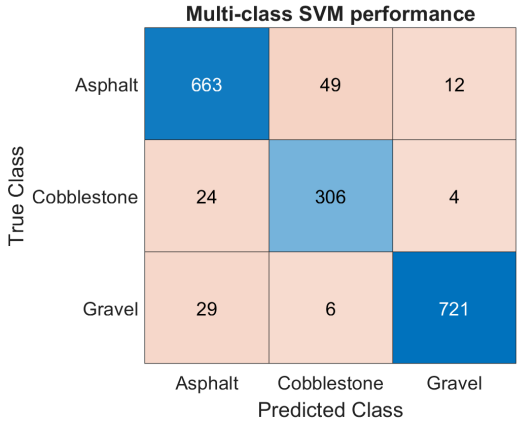


Figure 5.4: Confusion matrix on the multi-class SVM model

The performance is promising and already achieved an accuracy of 93.1% on validation data. Another test that could be done now is to investigate if the feature selection process done in section 5.1.3 has decreased the accuracy of the prediction. Using the same algorithm on the whole dataset (that has 39 features) the accuracy is 93.9%, is clear how those removed features are not significant and it has been a good choice to discard them to increase generalization and an easier computation.

Feature importance

A last interesting analysis that can be done on the multi-class SVM model is to understand the importance of the used features and which are more used during the classification. This kind of analysis is really important for some reasons in a machine-learning problem:

- By looking at the feature that the model uses it can be checked if the algorithm behavior is similar to how expected;
- When a prediction is made, the reasons for it can be explained by looking at the important features.
- An idea of which features perform better is indicated, and it could be useful for future development.
- It's easier to explain how the algorithm works to a client who is not an expert in the field.

The premises for this analysis are:

1. The SVM uses a linear kernel;
2. The input features are standardized before the model training;

These structural choices for the model allow us to compute a meaningful importance score for every feature that indicates how much these influence the predictions.
An SVM classifier is based on a division hyper-plane that, in the linear case, is represented by the coefficients $\boldsymbol{\beta}$.
This latter can be influenced by two factors:

1. **Feature scale:** If the feature values are on different scales, the coefficients are influenced by this and need to balance the difference by taking values not comparable to each other.
2. **Feature importance:** With a constant scale, a feature that affects more classification results will have a higher coefficient than the others.

Since the training dataset is standardized, the first factor is no longer relevant, and this means that the model coefficients are a valid representation of the feature importance.

In a multi-class scenario, every feature will have a coefficient for each binary learner, in this case, six values per column. This means that the ECOC model has a coefficient matrix where β_i^j represents the coefficient of the i-th feature on the j-th learner.

	Feature 1	**Feature 2**	**...**	**Feature 19**
Learner 1	β_1^1	β_2^1	...	β_{19}^1
Learner 2	β_1^2	β_2^2	...	β_{19}^2
Learner 3	β_1^3	β_2^3	...	β_{19}^3
Learner 4	β_1^4	β_2^4	...	β_{19}^4
Learner 5	β_1^5	β_2^5	...	β_{19}^5
Learner 6	β_1^6	β_2^6	...	β_{19}^6

Table 5.2: Coefficient matrix of the multi-class SVM.

The absolute value of β can be considered since at this moment is not interesting to know if the feature has a direct or inverse relationship, but only how much it influences the result.
Then these values are rescaled in the interval $[0; 1]$ on each row so that can represent a percentage of importance for the learner. For every row j, all the values are rescaled as:

$$z_i^j = \frac{\beta_i^j - \min(\boldsymbol{\beta^j})}{\max(\boldsymbol{\beta^j}) - \min(\boldsymbol{\beta^j})}$$

to compute the importance percentages z.

After these transformations, two visualizations of the feature's importance can be plotted.

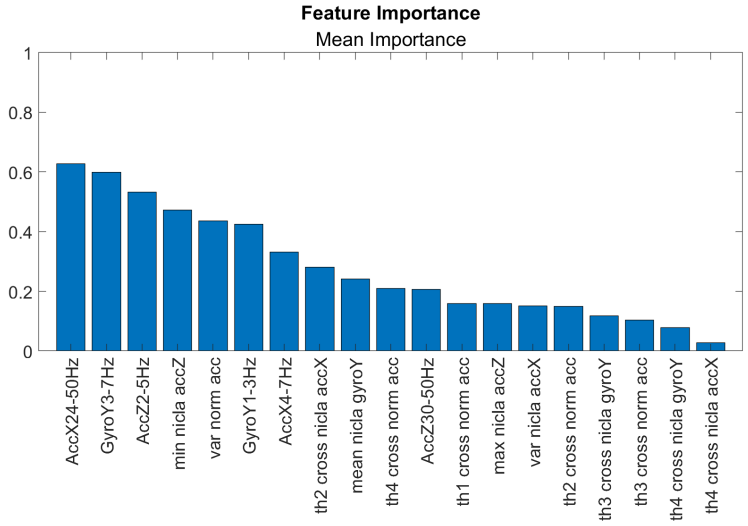


Figure 5.5: Mean feature importance

The percentages plotted above are the mean of the columns vector $\boldsymbol{z_i}$ and so the average importance of the feature by looking at all the six learners. The frequency domain features turn out to be very important together with the minimum value recorded on the Z-axis of the accelerometer and the variance of the norm of acceleration.

A more detailed analysis can be done by visualizing the feature's importance on each binary learner.

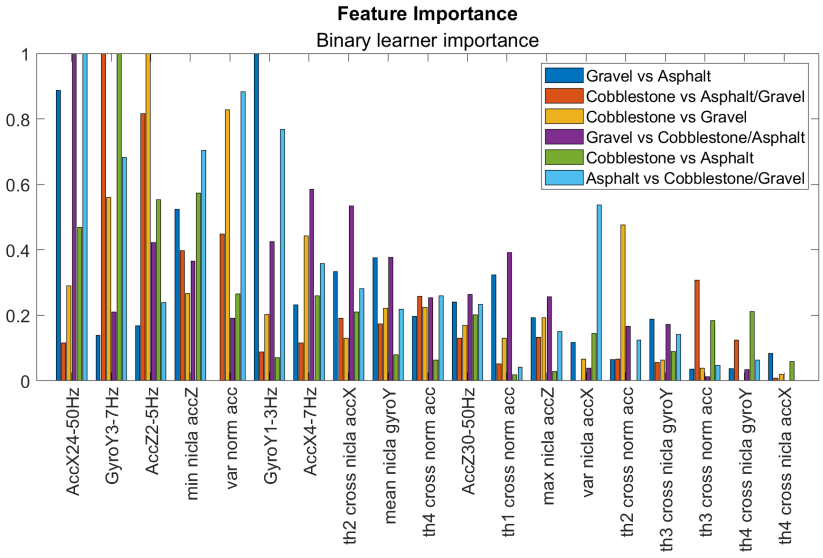


Figure 5.6: Feature importance on every binary learner

Analyzing this graph is clear how most of the features are not important for every binary classification, but each one is capable of characterizing a terrain class.

For example, the feature with the highest mean importance is the integral on $[24, 50]$Hz interval of the X-axis accelerometer frequency response.

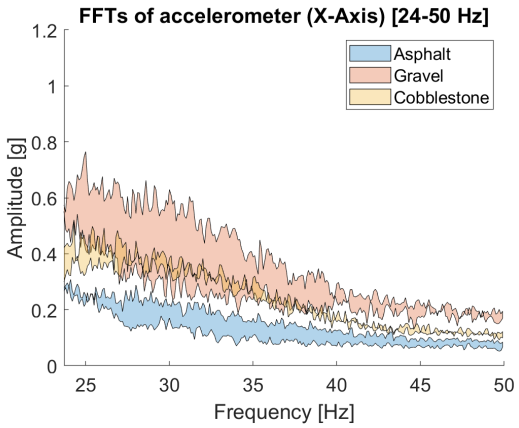


Figure 5.7: Frequency response of the three terrains signals on the X-axis of the accelerometer (zoomed on $[24, 50]$Hz range)

The feature importance graph shows that this variable is the most important in the classification between gravel and the rest of the dataset, asphalt and the rest of the dataset, and is also meaningful in the classification between gravel and asphalt. The visualization of the frequency of responses explains the reason for this importance.

Then the second most important feature is the same measurement as above computed on the Y-axis of the gyroscope in the $[3, 7]$Hz interval.

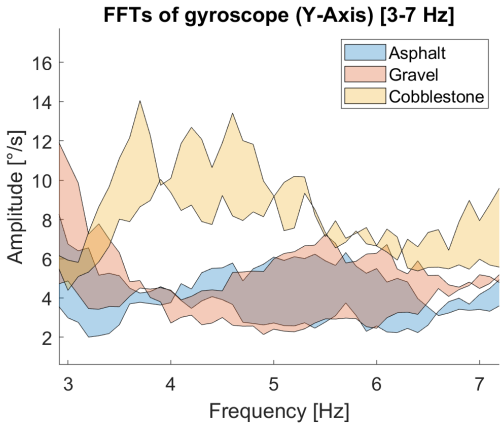


Figure 5.8: Frequency response of the three terrains signals on the Y-axis of the gyroscope (zoomed on $[3, 7]$Hz range)

The importance graph makes clear that this value is used to characterize the cobblestone ground and indeed, by looking at the frequency spectra the cobblestone

response separation is clear. The visual inspection also demonstrates why the feature is not used to distinguish asphalt and gravel.

The third and last feature that will be graphically visualized is the integral of the frequency response on the Z-axis of the accelerometer in the $[2, 5]$Hz interval.

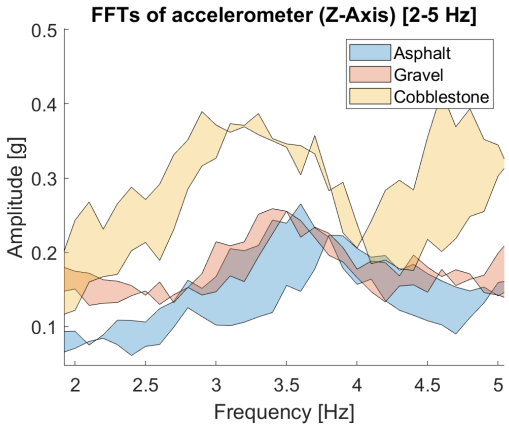


Figure 5.9: Frequency response of the three terrains signals on the Z-axis of the accelerometer (zoomed on $[2, 5]$Hz range)

Once again, this feature has high importance in distinguishing cobblestone samples from the other, and the frequency analysis graph confirms a difference between this latter terrain and the others.

5.2.2 Output modeling

An idea suggested by Brooks paper [9], and also considered when using RNN [12][13] is to exploit the fact that the output of this problem is a time-series which usually changes value slowly, in fact during a bike ride a typical use involves sporadic variation in terrain beneath. Until now the classification algorithm tried to predict the terrain experienced for every sample singularly, which is an approach that does not consider an important characteristic of the output.
What has been thought is to adding this information to the algorithm refining the predictions of the multi-class SVM using a Hidden Markov Model. The HMM is a Markov Model in which the observations depend on a hidden process that can not be observed. A more detailed explanation of how this algorithm works will be presented in appendix A.2.

To exploit this technique, the SVM predictions as been taken as the alphabet of observable symbols that are emitted by the current terrain beneath that represent the hidden state. By setting the transition matrix so that the probability of remaining in the same state is sufficiently higher than the probability of changing state, the predictions will be refined promoting consecutive identical states as output instead of having quick class changes.

The prediction enhancing is done using the Viterbi algorithm which will be further investigated in the appendix.

Offline test of prediction enhancing

A first test to understand if this approach can be useful for the problem has been done in an offline setting. Since the classes in the dataset are organized per class, it already has the shape described above, with the class keeping the same value for long periods, so is possible to implement and try the output correction using an HMM.
An HMM capable of modeling the SVM predictions will have the following structure:

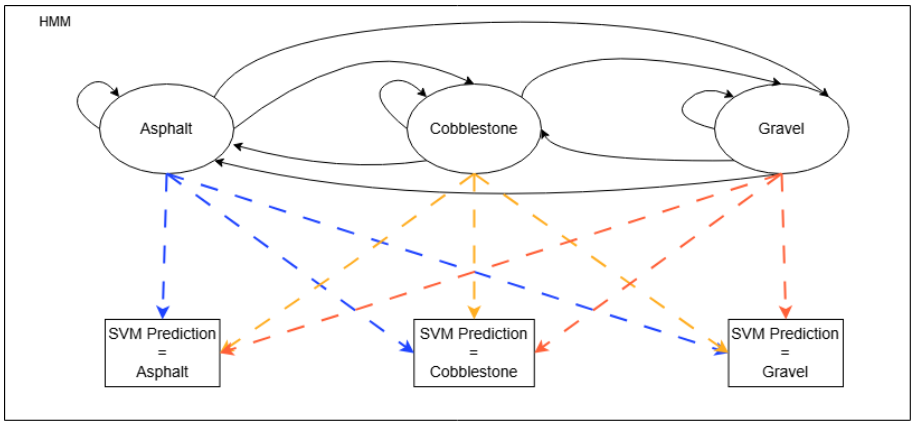


Figure 5.10: HMM considered during the prediction enhancing process. In the figure the three states (ellipsis) and the three emitted symbols (rectangles) are depicted. The solid lines represent the possible future steps for each state which will follow the probabilities of the transition matrix. Each state is then linked via a dashed lines with the symbols that can be observed (with probability controlled by the emission matrix) while in this. The dashed lines are colored blue, yellow, and red correspondingly for asphalt, cobblestone, and gravel

For this experiment, the transition and emission matrix have been chosen empirically. The transition matrix is set as:

$$A = \begin{bmatrix} 0.94 & 0.03 & 0.03 \\ 0.03 & 0.94 & 0.03 \\ 0.03 & 0.03 & 0.94 \end{bmatrix}$$

So it is symmetric and in every hidden state, there is the 94% of probability to remain and the 3% for each other state switching. The emission matrix used is:

$$B = \begin{bmatrix} 0.80 & 0.10 & 0.10 \\ 0.10 & 0.80 & 0.10 \\ 0.10 & 0.10 & 0.80 \end{bmatrix}$$

This means that while you are in a state there is an 80% probability to observe the exact prediction from the SVM, and there is a 10% possibility for both the kinds of

misclassifications.
The pipeline used for the offline prediction enhancement is represented in figure 5.11.

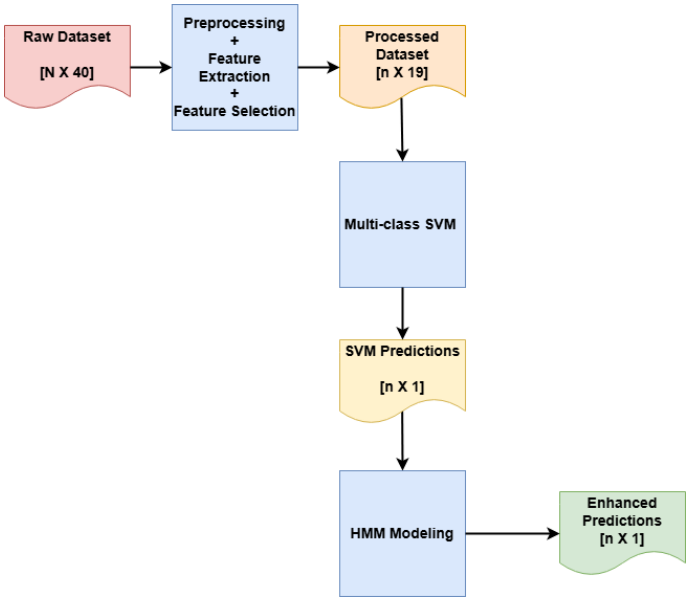


Figure 5.11: SVM classification with HMM output modeling pipeline

Offline performance with HMM predictions modeling

The same procedure described in section 5.2.1 has been applied and then the predictions provided by the SVM have been processed with the HMM to decide if this technique is valid.

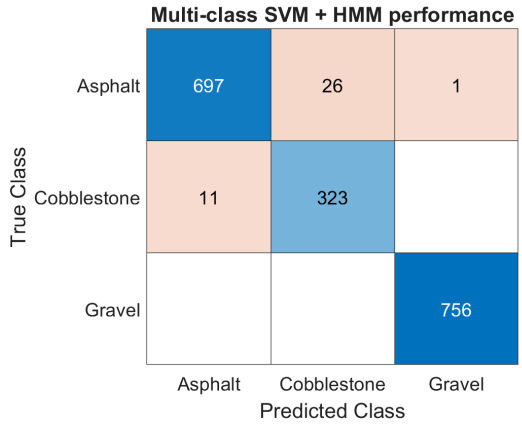


Figure 5.12: Confusion matrix on the multi-class SVM model and HMM output modeling

The accuracy has gone from 93.1% of the single multi-class SVM to 97.8% with the prediction enhancement. This result is promising and so will also used in the implementation of the online-classification algorithm.

5.3 Online classification problem

In this section, some refinement of the terrain recognition algorithm will be described, and after that, the performance in a real-world scenario will be evaluated.

5.3.1 Online simulation

Test dataset

To evaluate the recognition capability of the algorithm, a test dataset has to be collected. The main difference with respect to the previous measurement collections is that in this case, all the riding parameters will be varied in the same data acquisition, except for the damper status which is difficult to change during an experiment. The evaluation will be done in two scenarios:

1. E-Bike with the damper unlocked traversing a path starting on cobblestone, continuing on asphalt, and finishing on gravel.

2. E-bike with the shock absorber locked following the path described before but in the reversed direction, so starting from the gravel, reaching asphalt, and ending on cobblestone.

The riding pace has been simulated to be the most natural possible with accelerations, slowdowns, and even stops.

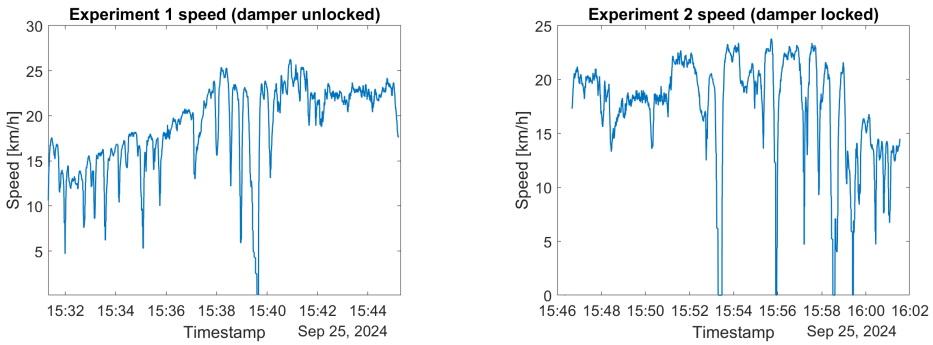


Figure 5.13: Speed recorded during the first (on the left) and the second (on the right) experiment

From the frequency analysis conducted before, speed is the external agent that influences in a more impactful way the model's features so it has been made to vary as much as possible.
Both experiments have a length of more than ten minutes distributed on the three terrains to emulate a normal urban bike ride.

Simulation algorithm

In order to have a test environment as close as possible to real use, we want to test the classifier iteratively, that is, not classifying the whole dataset at once, but simulating the consecutive arrival of new measurements.
This behavior is obtained thanks to a script that loads the dataset and immediately applies the `create_rev_groups` function to divide the raw signal into more single samples. From these samples immediately all the columns that do not refer to inertial measurements and bike speed are removed since these are not used during the classification. Each group of measurements is then analyzed singularly in a `for` loop where it follows all the steps to transform the raw sample to an instantaneous prediction.

An apposite function called `create_features`, capable of elaborating this kind of measurement has been created. The function applies the same identical preprocessing procedure done during the model training with some differences:

- Only the strictly necessary operations are done, and these have been studied to be time-efficient;
- Some limit cases have been covered since the final sample could have a low number of measurements;
- The algorithm took the whole dataset given as input and transformed it into a single row of features, so the input must have the correct dimension, set to be of three wheel revolutions during training and so also in the test case.

The features vector is elaborated following the classification logic that will explained in the following section, and then finally the resulting prediction is stored in a list that will be later analyzed to evaluate the performance.

5.3.2 Classification algorithm

As the first step, the online scenario simulator loads the SVM model that has been trained in the offline setting. Then begin to iteratively predict the class of the $i-th$ sample. Since the posterior probabilities function has also been trained during the model definition, for each iteration the probabilities of belonging to the three classes are computed.
The output prediction will be then enhanced using two techniques:

1. Output modeling with an HMM as seen during the offline analysis;
2. Class change prevention logic for samples that have low confidence based on the posterior probabilities of the SVM;

HMM online predictions modeling

The output-enhancing technique as seen in the offline setting is not feasible when predicting iteratively, because in this case we only have past predictions and the

new one instead of the complete series of predictions.

The first modification to this approach has been to apply the Viterbi algorithm only to a subset of predictions. The length of this vector is a new parameter of the algorithm that can be chosen empirically and according to hardware needs. A bigger dimension of the vector will have a wider vision of the past predicted terrains but is more demanding for both memory and execution time.

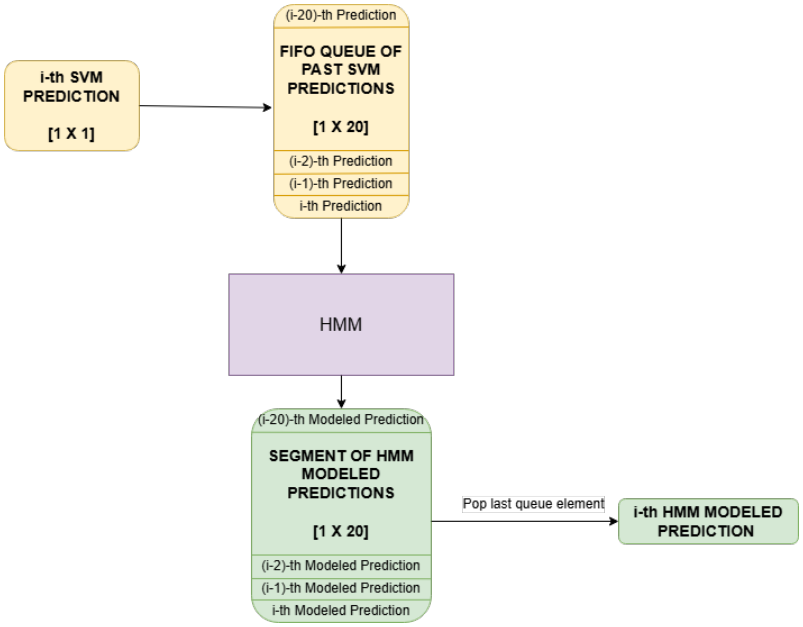


Figure 5.14: Diagram of HMM online modeling process

While tuning this parameter it has been noticed that after a certain length, increasing the number of past predictions stored gives no advantages.
Is important to notice that the HMM will always model a set of SVM predictions, and not the segment modeled in the previous step so that every iteration is independent from the previous one and no previous HMM errors will be propagated in the next classifications.

The transition matrix of the HMM has been empirically chosen as in the offline setting. The choice for the online algorithm is:

$$A = \begin{bmatrix} 0.92 & 0.04 & 0.04 \\ 0.04 & 0.92 & 0.04 \\ 0.04 & 0.04 & 0.92 \end{bmatrix}$$

Similar to the one done in section 5.2.2 so that the model will prefer a stable behavior of the predictions.

Regarding the emission matrix, a dynamic approach has been implemented. Since the emission matrix explains the probability of observing a symbol while in a certain state, it can be varied according to the posterior probability. The value used to

model the matrix is the mean of a certain number of previous posterior probabilities for the predicted class. This value ($\bar{P}(y_{pred}|\boldsymbol{x_{pred}})$) will be better explained below.

$$B = \begin{bmatrix} \bar{P}(y_{pred}|x_{pred}) & \frac{1-\bar{P}(y_{pred}|x_{pred})}{2} & \frac{1-\bar{P}(y_{pred}|x_{pred})}{2} \\ \frac{1-\bar{P}(y_{pred}|x_{pred})}{2} & \bar{P}(y_{pred}|x_{pred}) & \frac{1-\bar{P}(y_{pred}|x_{pred})}{2} \\ \frac{1-\bar{P}(y_{pred}|x_{pred})}{2} & \frac{1-\bar{P}(y_{pred}|x_{pred})}{2} & \bar{P}(y_{pred}|x_{pred}) \end{bmatrix}$$

Doing so, when the posterior probability of the chosen class computed by the SVM is low, then also for the HMM there will be a reduction of probability for the current state to emit the correct prediction. In the opposite situation, if the SVM has a high probability of being correct, the HMM will trust more that the symbol is coming from the correct state.

State change prevention for uncertain predictions

An additional logic that considers the posterior probabilities returned by the SVM has been implemented. The posterior probabilities come as a vector of three percentages that estimate the possibility that a sample belongs to each of the three terrains. Considering y a specific observation between the classes and $\boldsymbol{x}$ the coordinates of the sample (the feature values) in the hyper-space, the SVM will return three:

$$P(y|\boldsymbol{x}) = P(y) \text{ in point } \boldsymbol{x} \text{ of the hyper-space.}$$

one for each value that y can take.

Since it could be hard to visualize this value on a high-dimension hyper-space, an external example that only uses two dimensions will be presented. This example refers to flower species classifications using the length and the width as features.

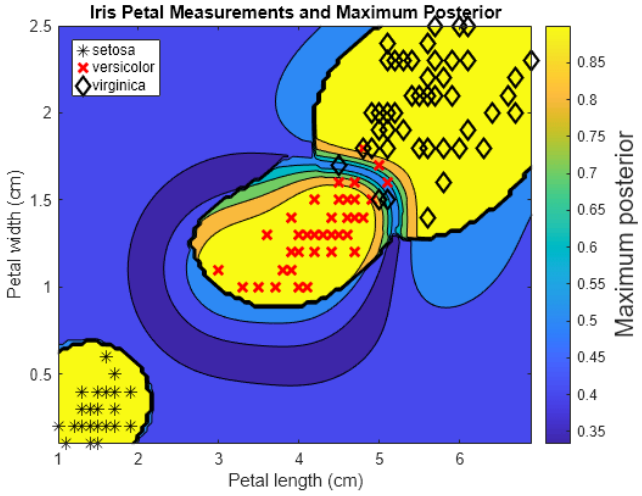


Figure 5.15: Example of posterior probabilities assignment on a low-dimension classification problem

In the graph, only the maximum value between the probabilities has been represented by the background color. On two dimensions is easier to visualize how this model can assign these probabilities to each class relying on the distance from the decision boundaries.

The general idea behind the logic implemented in this phase is to not only look at the predicted ground but also at the probability of belonging to it. Analyzing the certainty of the prediction it can be chosen if this is reliable enough to be accepted, otherwise in order to avoid state changes caused by unreliable predictions, the previous state is kept.
Another consideration that has been made is that sometimes the SVM model makes mistakes with high posterior probability, but usually, these are single and rare events. Because of this, the posterior probabilities have been smoothed by considering the mean of these in the last n instants. These three values at instant t are:

$$\bar{P}_t(y|\boldsymbol{x}) = \frac{1}{n} \sum_{i=t-n}^{t} P_i(y|\boldsymbol{x})$$

for each y in the classification problem.
This procedure introduces another parameter that is strictly linked to the responsiveness of the algorithm to terrain changes.

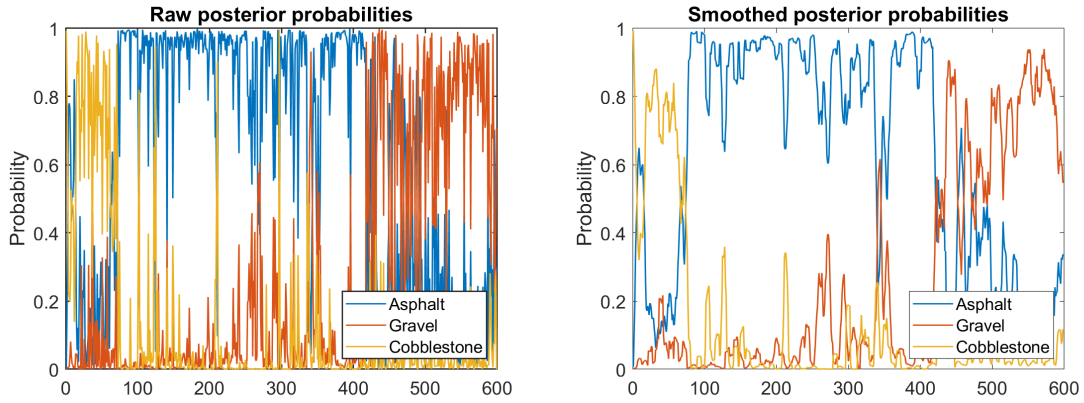


Figure 5.16: Comparison of an example of raw (on the left) and smoothed (on the right) SVM posterior probabilities

This parameter will need to be tuned to ensure a good trade-off between the smoothness of the response posterior probabilities and responsiveness during terrain switches.

Of the three means computed, the one corresponding to the predicted class is taken. When this value is low, the algorithm is not sure of the prediction, while high mean probability values are possible only if the model is consistently sure of the predicted class in a certain considered time interval.
As a design choice for the algorithm, a stable prediction behavior is preferred so that the output displayed to the user does not change rapidly. To ensure this the algorithm will reject new predictions if the certainty of those does not reach a threshold, that will be a parameter of the algorithm.

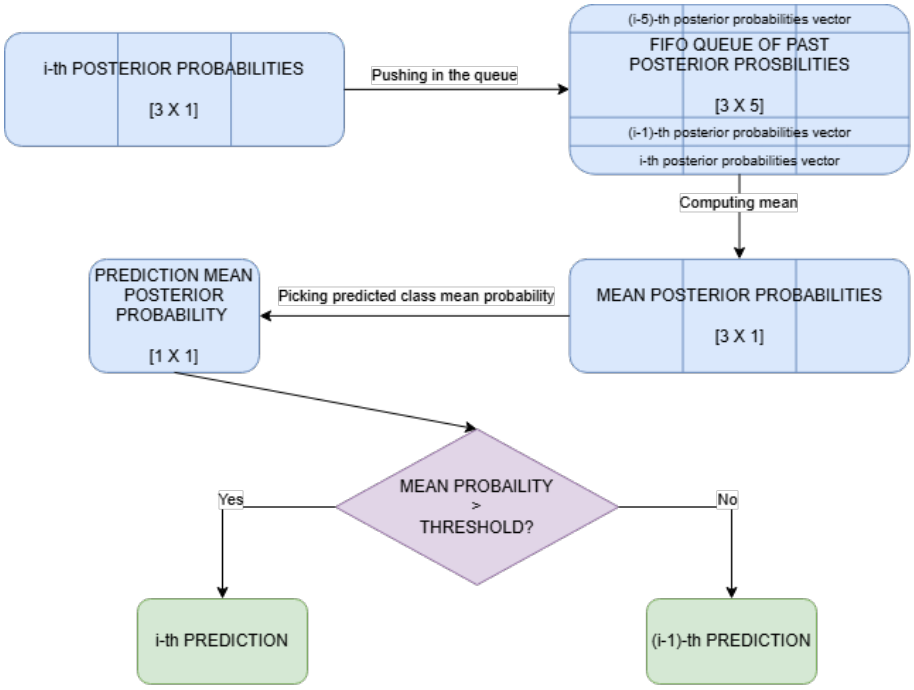


Figure 5.17: Logic to choose if picking the new prediction or keep the last one

Complete terrain recognition algorithm

The full algorithm will be a merge of the following phases:

1. **Preprocessing:** The raw sample is processed to have a vector of features representing the current terrain beneath;
2. **SVM Classification:** Using an SVM a class is predicted and the posterior probabilities are computed;
3. **HMM Output Modeling:** An HMM enhances the output considering the last prediction done by the SVM;
4. **Unreliable prediction rejection:** Using a logic that takes the posterior probabilities as input, it is decided if the current prediction can be kept or if it is better to reject it.

The algorithm can be controlled using the three parameters in table 5.3.

Name[1]	Description	Symbol[2]
HMM_SEQUENCE_LENGTH	Number of previous predictions considered by HMM	h
MEAN_POSTERIOR_WINDOW	Number of previous posterior probabilities considered to compute mean posterior probability	m
MIN_PROBABILITY_THRESHOLD	Minimum mean posterior probability to accept model's prediction	p

Table 5.3: Algorithm parameters description

These parameters will modify the behavior of the model. In particular:

- **HMM_SEQUENCE_LENGTH:** It has been found that after finding a sufficient context window, further increasing this parameter gives no benefit, and does not produce changes in the predictions.
- **MEAN_POSTERIOR_WINDOW:** Practically, this parameter changes the responsiveness of the model, a high value will correspond to a slower transition from one class to another, while low values make the algorithm less robust to interferences and misclassification of the SVM.
- **MIN_PROBABILITY_THRESHOLD:** Affect the model behavior similarly to the previous parameter. A high threshold will need more certainty in the last observations to switch classes, which means that it will be slower, while a low value could make too easy the change among the classes, with a higher risk of misclassifying.

These hyper-parameters have been tuned and these are the resulting values:

Parameter Name	Value
HMM_SEQUENCE_LENGTH	20
MEAN_POSTERIOR_WINDOW	5
MIN_PROBABILITY_THRESHOLD	0.75

Table 5.4: Tuned values used in this work

Since the HMM value is not that influent, and the smoothing of the posterior probabilities is needed for the algorithm to work properly, the `MIN_PROBABILITY_THRESHOLD` could be exposed to the user to command the behavior of the software.

[1]The name used in the MATLAB script
[2]Symbol that will be used in the graphical representation

The full algorithm is graphically represented in figure 5.18.

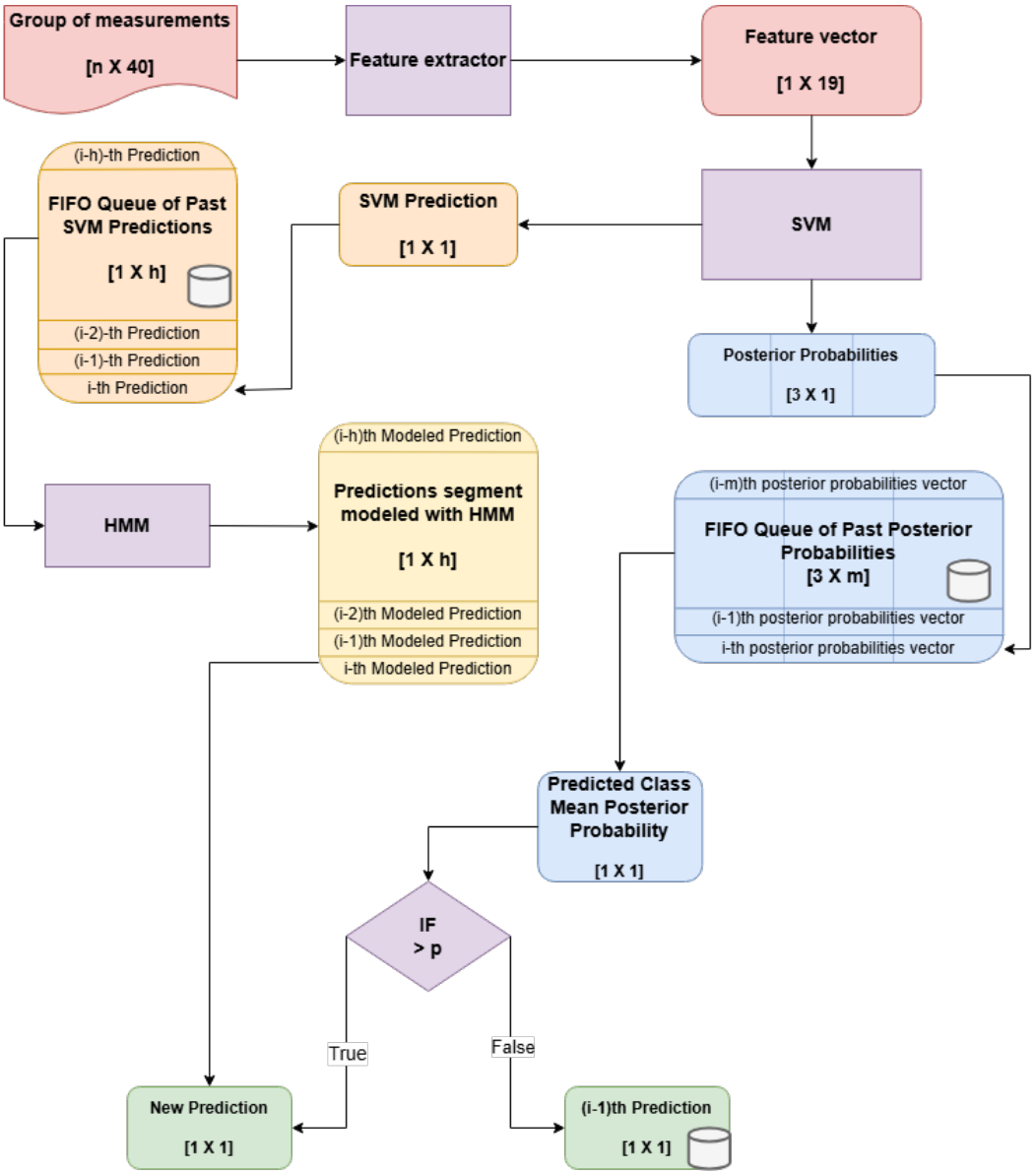


Figure 5.18: Graphical representation of the terrain recognition algorithm

In this chart:

- The purple components are processes that elaborate data;
- The red, orange, yellow, and green blocks represent the steps the data go through to be transformed from raw measurements to the final prediction;
- The auxiliary information, like the posterior probabilities, are depicted in blue;
- When a block contains the database symbol, it means that the value must be stored by the algorithm during the iterations.

5.3.3 Performance of the online classification algorithm

The performance of the algorithm presented in section 5.3 will be evaluated on the test datasets (5.3.1).

Test Experiment 1 (damper unlocked)

The SVM alone is already capable of good classification, reaching an accuracy of 87%. In figure 5.19 it can be seen:

- The real labels represented by the background color of the plot;
- The predicted terrain in every instant of the experiment looking at the blue bars;
- The three smoothed prediction probabilities each one with a differently colored line.

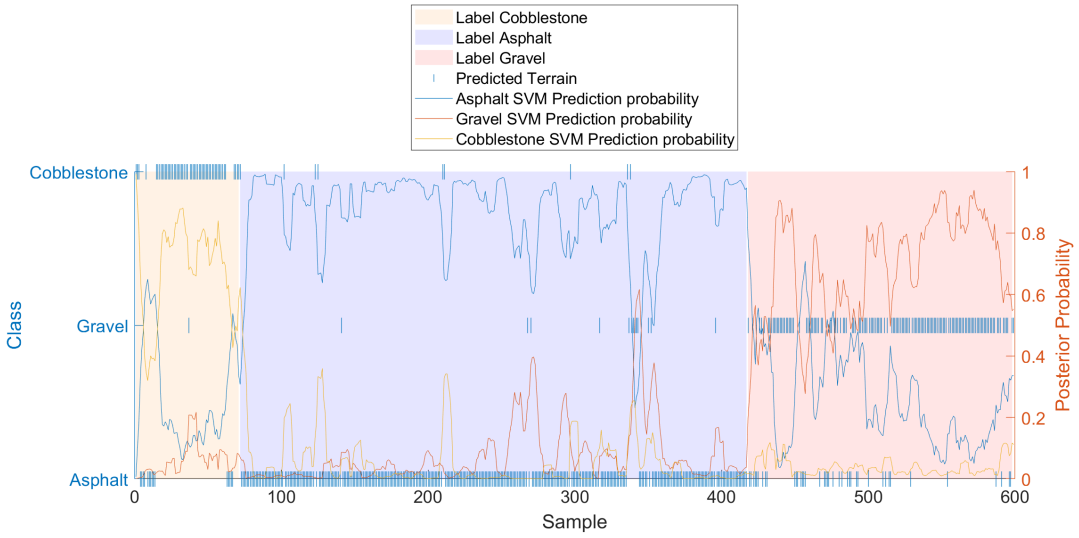


Figure 5.19: SVM Performance on Test Dataset 1

Looking at the graph is clear how many errors are committed just because of the unstable behavior of a classifier that predicts every instant without context.

The simulation has been run again but with the whole algorithm described previously in the chapter, and the same chart has been plotted. The accuracy is increased up to 96% and a smoother output has been produced.

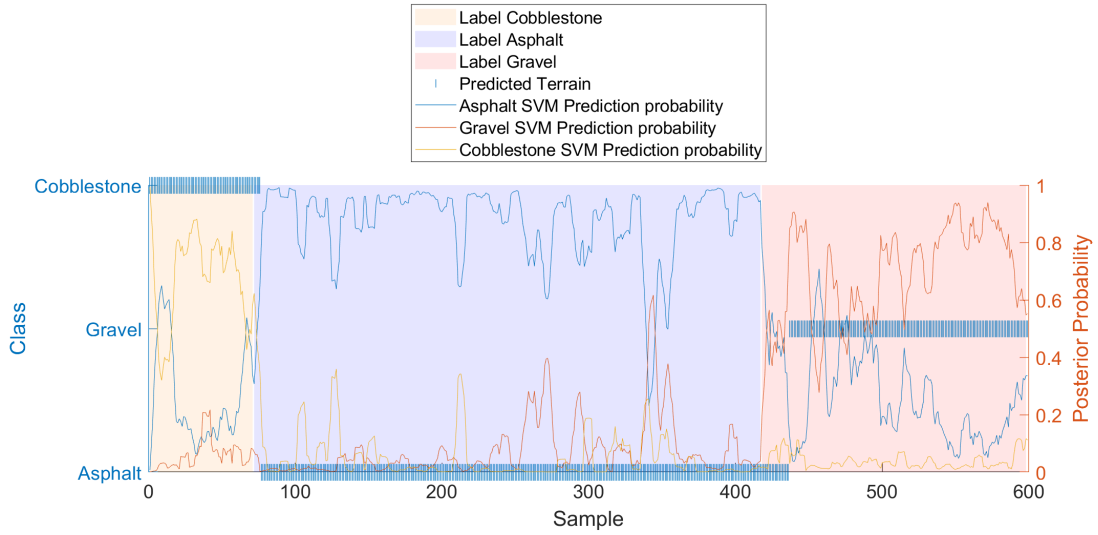


Figure 5.20: Complete Recognition Algorithm performance on Test Dataset 1

The target of having a stable output value is accomplished and, in this case, all the errors committed by the algorithm are in the change between classes.
In particular, looking at the confusion matrix, the reason for the errors can be investigated.

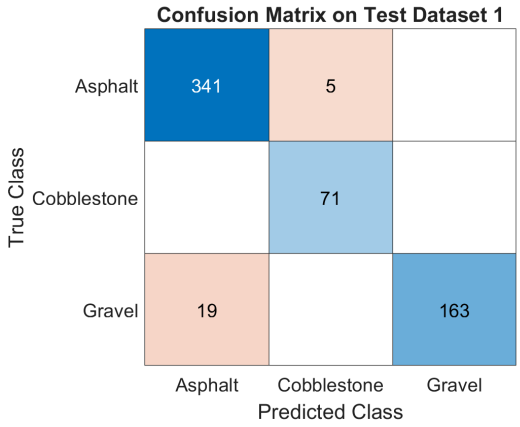


Figure 5.21: Confusion matrix of the prediction on Test Dataset 1

The model makes five mistakes in the first switch and nineteen in the second.

- The errors on the first switch were expected since the algorithm introduces a delay in the predictions;

- The second switch seems more problematic, and the reasons for this can be found by analyzing the location where it happens. In figure 5.22 the terrain experienced during the change of classes can be seen. This location that has been labeled as gravel is pretty different from the ground observed during training, and by also looking at the posterior probabilities is clear that the SVM model is struggling to make a choice.

Figure 5.22: Terrain in the location where the model does most of the errors

Test Experiment 2 (damper locked)

The same kind of analysis can be done on the second test dataset, so we can see if the shock absorber has a significant effect on the performance of the model.

As already done before, the performance on the SVM alone is computed so that we have a baseline to use as a comparison.

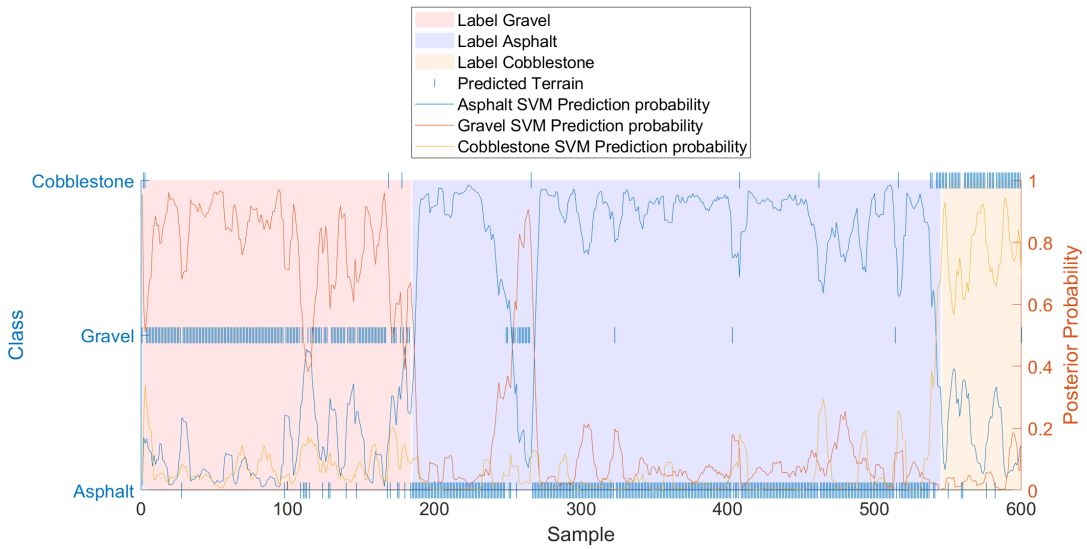


Figure 5.23: SVM Performance on Test Dataset 2

The accuracy is higher compared to the one achieved with SVM in the first test experiment, which is 91% in this case. Also, the mean posterior probabilities are more stable. These differences are likely to be given by the fact that the shock absorber, which is unlocked in the first experiment, tends to hide the footprint of the various terrains, making the measurements collected in this scenario harder to distinguish.

After observing how the SVM alone performs, the dataset is fed to the complete terrain recognition algorithm, looking for an accuracy increase.

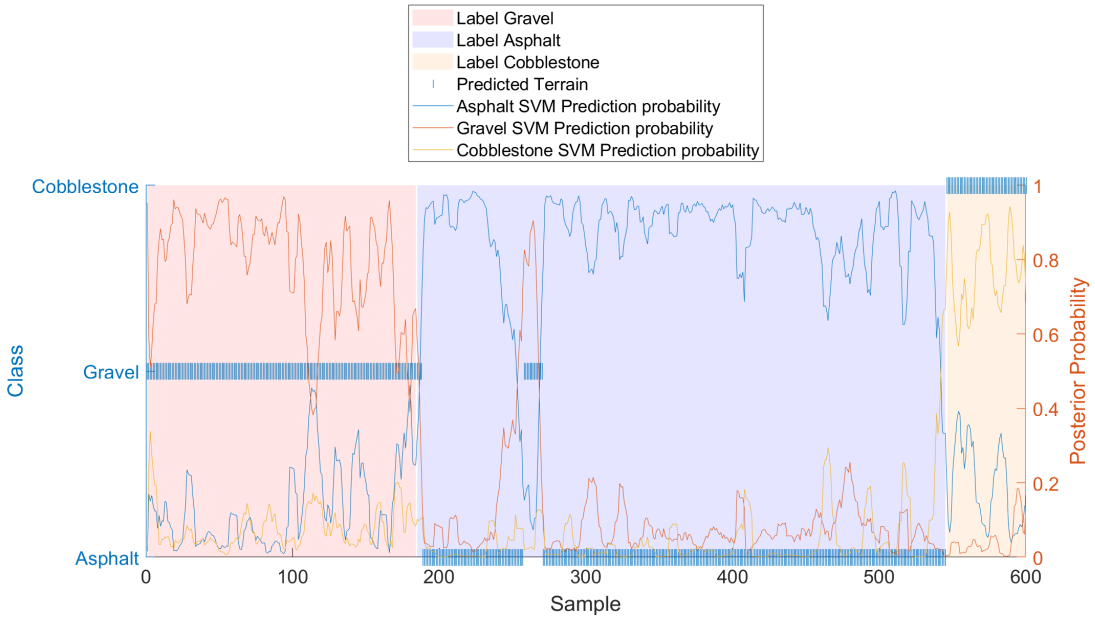


Figure 5.24: Complete Recognition Algorithm performance on Test Dataset 2

Also in this second experiment, the procedures implemented in addition to the SVM give a boost to the predictions' correctness, scoring 97% in total accuracy, and the same stable behavior is observed.
Computing the confusion matrix for this simulation some observation can be done. In this simulation, all the errors are done on asphalt, misclassifying this ground with gravel. However, these have been done in two distinct situations:

- Four errors are committed during the change of class, as already commented for the first experiment, a delay is expected because of the algorithm structure;
- The second moment of misclassification happens in the middle of the asphalt path. Thanks to the GPS signal the location has been found and is depicted in figure 5.26.

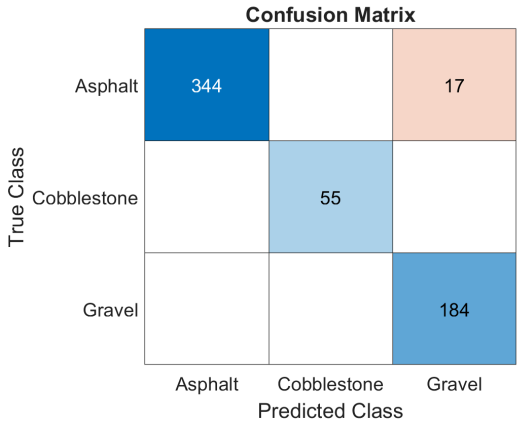


Figure 5.25: Confusion matrix of the prediction on Test Dataset 2

Figure 5.26: Ground beneath in the location where the model does most of the errors

The image shows two factors that badly influence the algorithm precision:

1. The path is uphill, and since there is no orientation method in the software, the measurements will be different due to the rotated reference system;

2. The asphalt is clearly damaged, so the measurements are likely to be similar to those collected on the gravel.

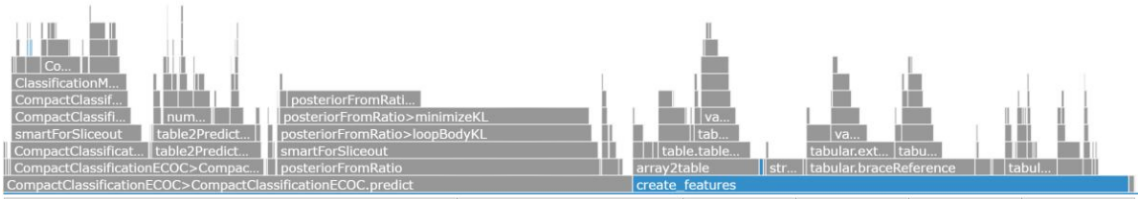
Prediction Time analysis

A final inspection that has been done for the algorithm is complexity analysis and execution time estimation.

The time complexity of a single prediction is constant ($O(1)$) since there are no loops or variations in execution based on the input. The only way to have a variation in runtime is to change the structure of the algorithm. In particular, if you decide to train a new model that takes a different number of wheel revolutions as input, the change in the length of the measurements matrix will affect the execution time. As the number of observations per sample increases, the feature extraction function will have to handle larger tables, so operations such as multiplication between matrices, which is necessary for rotation of the inertial measurements reference frame, will

take longer.
The space complexity of the algorithm is only influenced by two parameters, which are `HMM_SEQUENCE_LENGTH` and `MEAN_POSTERIOR_WINDOW`, that control the length of the stored arrays, in addition to a single previous prediction, thus being constant.

The execution time has been computed by measuring the time it took for the simulation to run and dividing it by the number of predictions made. The simulation takes 24.4 seconds to run for an input containing 600 samples, which means that the algorithm takes about 0.04 seconds per prediction[3].
Finally, to understand which part of the algorithm influences the most the execution time, a profiling analysis has been conducted. Using the MATLAB profiling function the result in figure 5.27 has been observed.



Function Name	Function Type	Calls	Total Time (s)	% Time	Time Plot
CompactClassificationECOC>CompactClassificationECOC.predict	Class method	600	13.562	55.6%	
create_features	Function	600	10.580	43.4%	
hmmviterbi	Function	600	0.072	0.3%	
mean	Function	600	0.026	0.1%	
height	Function	1	0.000	0.0%	
Self time (built-ins, overhead, etc.)			0.142	0.6%	
Totals			24.382	100%	

Figure 5.27: Recognition Algorithm runtime profile

Almost all of the execution time is spent in the SVM classification and the feature extractor function. Since the first is a class method, it can not be changed.
However, the `create_features` function can be further analyzed, looking for possible optimizations. A high percentage of its runtime is spent in table manipulation, in fact during all the work this data structure has been used because makes the code more readable, and makes it easier to avoid errors while manipulated. If an improvement in the efficiency of the algorithm is needed, the entire code could be restructured so that it uses only matrices instead of tables.

[3]Execution time on an Intel Core i5-7300U

Chapter 6

Conclusions

In this last chapter, the results obtained from this work will be presented and compared to the goals set during the preliminary stages of the project. Finally, a list of possible future developments to the algorithm will be described.

6.1 Project results

As explained in this document introduction, the general goal of the project is to solve the terrain recognition task. The latter can be decomposed into two parts:

Prediction accuracy

As a primary objective, the algorithm must be capable of returning accurate predictions of the ground beneath. This functionality has been achieved thanks to the following stages:

- Collection of labeled datasets in all the possible riding conditions;
- Processing and analysis of the collected measurements;
- Characterization of the terrains by extraction of meaningful features;
- Training of a machine learning algorithm capable of distinguishing the classes from the features given as input.

The test case results confirm that sufficient accuracy has been achieved.

Test case	Damper	Duration	Number of samples	Accuracy
Experiment 1	Unlocked	13min 54s	599	96%
Experiment 2	Locked	14min 53s	600	97%

Table 6.1: Encoding matrix of ternary-complete design.

Output stability

In addition to accuracy, another important feature is the stability of the returned output. This goal was also met, and it is demonstrated by figure 5.20 and 5.24.
Looking at the estimations pattern, it is confirmed that the implemented output modeling techniques avoid fast changes during the classification, at the cost of slightly reducing the responsiveness of the algorithm.
This procedure introduces a variation in model evaluation. Usually, the performance of a model is judged by the accuracy achieved, however with a stable output, is more meaningful to analyze how many events (considered as a set of near samples) are misclassified and the reason for that.

6.2 Future developments

During the course of the work, some potential improvements and additions to the algorithm were identified.

Improving train dataset

Collecting additional data for training would improve the algorithm. In particular, the dataset used to characterize the cobblestone ground is smaller than the other two, and it also contains less precise measurements due to the location exploited for the acquisitions which is unsuitable for an experiment.
The collection of new data can also be done to introduce new classes to extend the range of scenarios in which the algorithm can correctly operate.

Riding factors not considered

In chapter 3.2 a list of riding parameters is presented. In addition, some other factors have not been considered making the algorithm less robust when these are varied. In particular, all the experiments considered for the work have constant tire pressure and weight of the rider. A change in these values is likely to affect the collected inertial measurements, since a lower tire pressure can introduce more damping, and the user weight can empathize or reduce the intensity of the response.
Again, the solution is to increase the training dataset to have a more robust classification algorithm.

Shock absorber sensorization

The configuration of the shock absorber is the only considered riding factor that is not measured. It would be useful to add a sensor capable of storing this information in the dataset. It has been seen that the interaction with terrain is less intense when the damper is unlocked, so the performance could be improved if this feature is included.

Orientation algorithm

A weakness of the algorithm was observed during the testing phase. In fact, the use of a fixed reference frame for data representation limits accuracy when the bicycle is tilted, for example, due to an ascent or descent.
To solve this problem, an orientation estimation software can be implemented using IMU measures. Then, an appropriate rotation can be applied in real-time to the collected data so that the reference frame will always be the same.

Fusion of vision and vibration-based approach

Another technique found in the literature that can be introduced to enhance the performance is fusing the current vibration-based classification with a machine-vision algorithm [17]. The BRICK is already connected to a camera, but the collected images are not yet considered in the scope of this work.

Algorithm implementation on BRICK

Finally, the algorithm will have to be implemented on the e-bike prototype, plausibly integrating it into the BRICK. With the algorithm running on the BRICK, it will be easy to store the predictions and also display them to the rider in real-time.

Appendix A

Theoretical Background

A.1 Support Vector Machine

The SVMs are supervised learning models that, representing the input samples as points in the feature space, establish a boundary capable of dividing them according to their class into two different subspaces.

A.1.1 Linear case

In the linear case, the boundary is a linear classifier:

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 \tag{A.1}$$

The model also defines two "margins" :

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 = \pm 1 \tag{A.2}$$

It's possible to represent a generic margin with the values $\{1, -1\}$ since even with different values, the problem can be rescaled to come back to this situation leaving it unchanged.
The margin width maximization is important to define robust classifiers because it will guarantee a better generalization. A visual representation of the difference between large and narrow margins is depicted in figure A.1. In this image, two classifiers are represented, and both of them contain five points in their margins. However, the solid-line classifier has way larger margins compared to the other one and this will result in better generalization to future observations since it's further away from both the highly populated areas.

This width can be computed as double the distance between one of the margins and the boundary. This latter is given by:

$$d = \frac{|\boldsymbol{w}\boldsymbol{x} + w_0|}{||\boldsymbol{w}||} = \frac{1}{||\boldsymbol{w}||} \tag{A.3}$$

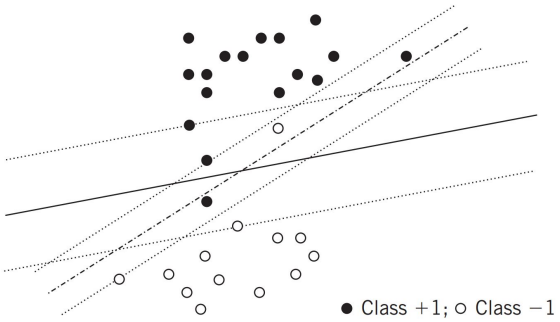


Figure A.1: Two SVMs for binary classification problem

A perfect separation of the two classes is not always achievable, so the concept of "soft margin" is introduced, which allows some points to be on the wrong side of the classifier.
The loss function associated with this model will be affected by both the points that lie on the wrong side of the space and the ones inside the area determined by the two margins, even if these are in the correct space side. These data points are called *support vectors*.

The optimization problem associated with the training of an SVM will consider both the maximization of the margin width and the minimization of support vector errors.
The width maximization problem can be transformed into a minimization one:

$$\text{Maximize} \quad \frac{1}{||\boldsymbol{w}||} \quad \rightarrow \quad \text{Minimize} \quad \frac{1}{2}||\boldsymbol{w}||^2 \tag{A.4}$$

where the $\frac{1}{2}$ term has been introduced to simplify future computations.

This discussion can be mathematically expressed with the following formulation. Given a set of training samples x_i with the respective labels $t_i \in \{-1, 1\}$, $i = \{1, 2, 3, ..., N\}$, compute the hyperplane (A.1) that:

$$\text{Minimize} \quad J(w, w_0, \epsilon) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} \epsilon_i \tag{A.5}$$

$$\text{Subject to} \quad t_n(w^T x_i + w_0) \geq 1 - \epsilon_i, \tag{A.6}$$

$$\epsilon_i \geq 0 \tag{A.7}$$

where ϵ_i are nonnegative errors. In particular, these values will be zero for points correctly classified and positive for misclassified or inside the margins samples.

The C parameter can be controlled by the user to set the penalty applied to misclassified or between the margin samples. By increasing it the margins will be closer reducing the generalization, while a low value will make the model less severe, allowing more errors.

Optimization problem

The optimization problem can be solved with the Lagrangian multiplier method, that is, finding the intersection of the constraint functions in a tangent point to the minimization problem function (so with derivative = 0).
The corresponding Lagrangian is given by

$$L(\boldsymbol{w}, w_o, \boldsymbol{a}) = \frac{1}{2}||\boldsymbol{w}|| + C\sum_{i=1}^{N}\epsilon_i - \sum_{i=1}^{N} a_i\{t_i y(\boldsymbol{x}_i) - 1 + \epsilon_i\} - \sum_{i=1}^{N}\mu_i\epsilon_i \tag{A.8}$$

where a_i and μ_i are Lagrangian multipliers.
The set of conditions associated with the problem is

$$a_i \geq 0 \tag{A.9}$$
$$t_i y(\boldsymbol{x}_i) - 1 + \epsilon_i \geq 0 \tag{A.10}$$
$$a_i(t_i y(\boldsymbol{x}_i) - 1 + \epsilon_i) = 0 \tag{A.11}$$
$$\mu_i \geq 0 \tag{A.12}$$
$$\epsilon_i \geq 0 \tag{A.13}$$
$$\mu_i\epsilon_i = 0 \tag{A.14}$$

Using the definition A.1 of $y(\boldsymbol{x})$ the optimization of $\boldsymbol{w}$, w_0, ϵ_i can be done.

$$\frac{\delta L}{\delta \boldsymbol{w}} = 0 \quad \Longrightarrow \quad \boldsymbol{w} = \sum_{i=1}^{N} a_i t_i \boldsymbol{x_i} \tag{A.15}$$
$$\frac{\delta L}{\delta w_0} = 0 \quad \Longrightarrow \quad \sum_{i=1}^{N} a_i t_i = 0 \tag{A.16}$$
$$\frac{\delta L}{\delta \epsilon_i} = 0 \quad \Longrightarrow \quad a_i = C - \mu_i \tag{A.17}$$

these result are then used to remove the $\boldsymbol{w}$, w_0, ϵ_i terms from the Lagrangian

$$\tilde{L}(\boldsymbol{a}) = \sum_{i=1}^{N} a_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j t_i t_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j}) \tag{A.18}$$

obtaining the dual Lagrangian.
The latter is subject to two constraints. The first is:

$$\sum_{i=1}^{N} a_i t_i = 0 \tag{A.19}$$

and the second, since $a_n \geq 0$, $\mu_i \geq 0$, and (A.17):

$$0 \leq a_i \leq C \tag{A.20}$$

Then, the maximization of the dual problem, which represents the SVM training phase, is done using an iterative routine. The most known are the Iterative Single Data Algorithm (ISDA [18]), and the Sequential Minimal Optimization (SMO [19]).

The predictive model is found by substituting in A.1 the weights found in A.15

$$y(\boldsymbol{x}) = \sum_{i=1}^{N} a_i t_i (\boldsymbol{x} \cdot \boldsymbol{x_i}) + w_0 \tag{A.21}$$

The solution can be interpreted as:

- The data points with $a_i = 0$ do not contribute to the predictive model since are not *support vectors*;
- The subset with $0 < a_i < C$, which implies $\mu_i > 0$ and so from A.14 $\epsilon_i = 0$, lie on the margin;
- The remaining samples, with $a_i = C$, lie inside the margin and can be either correctly classified or misclassified.

Finally, the process to determine w_0 is based on the *support vectors* with $0 < a_i < C$ so those in which $t_i y(\boldsymbol{x_i}) = 1$. Using A.21:

$$t_i \left(\sum_{j \in S} a_j t_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j}) + w_0 \right) = 1 \tag{A.22}$$

where S is the set of indices of the support vectors.

The most stable solution to solve this equation is given by multiplying both sides by t_i, exploiting the fact that $t_i^2 = 1$, and then averaging over all support vectors with $0 < a_i < C$ and solving for w_0:

$$w_0 = \frac{1}{N_M} \sum_{i \in M} \left(t_i - \sum_{j \in S} a_j t_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j}) \right) \tag{A.23}$$

where M is the subset of indexes of points having $0 < a_i < C$.

A.1.2 Nonlinear extension

The classifier described in A.1.1 is a valid technique, but it needs a strong assumption: the two clusters of data have to be linearly separable. This means that the two sets of data with different labels must be able to be divided using a linear boundary.

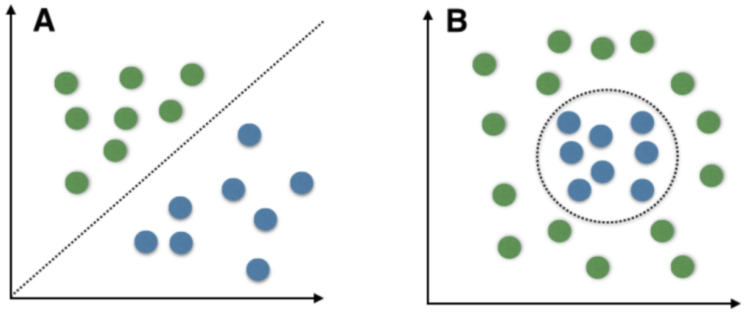


Figure A.2: Comparison between linear (A) and nonlinear (B) cases.

To overcome the problem, is possible to map the features vector to a higher-dimensional space, hoping that these will be linearly separable in the latter.

$$\boldsymbol{x} \quad \rightarrow \quad \phi(\boldsymbol{x}) \tag{A.24}$$

The mapping function should be chosen so that the inner product between two images $(\phi(\boldsymbol{x_1}), \phi(\boldsymbol{x_2}))$ of $(\boldsymbol{x_1}, \boldsymbol{x_2})$ points is given by:

$$< \phi(\boldsymbol{x_1}), \phi(\boldsymbol{x_2}) >= k(\boldsymbol{x_1}, \boldsymbol{x_2}) \tag{A.25}$$

where $k(\cdot, \cdot)$ is known as the kernel function. This means that the inner product in the higher-dimensional space can be computed as the kernel function in the original space.

The solution found before (A.21) is transformed as

$$y(\boldsymbol{x}) = \sum_{i=1}^{N} a_i t_i k(\boldsymbol{x}, \boldsymbol{x_i}) + w_0 \tag{A.26}$$

allowing the extension to nonlinear problems.

Two kinds of kernel often used are the radial basis function (RBF)

$$k(\boldsymbol{x}, \boldsymbol{y}) = exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{y}||^2}{\sigma^2}\right) \tag{A.27}$$

that can be tuned by changing the value of σ^2, and the polynomial function

$$k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y} + b)^n \tag{A.28}$$

which is controlled by the b and n user-defined parameters.

A.2 Hidden Markov Model

The HMM model is a Markov Model with a set of hidden states that emits observation from a known alphabet of symbols.

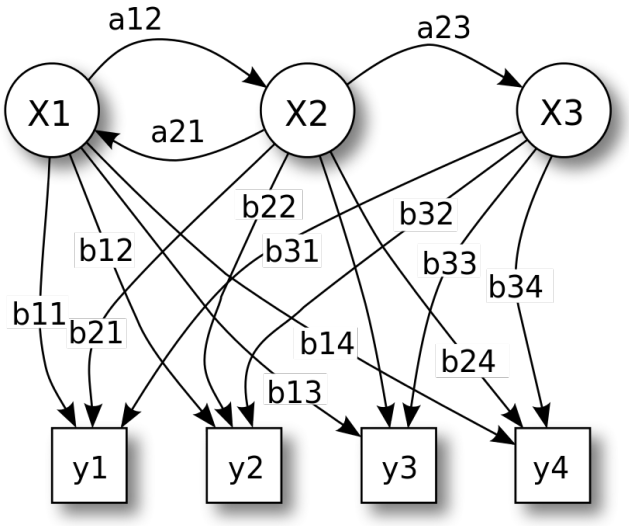


Figure A.3: Example of Hidden Markov Model

Every model contains a finite number of states s that are not observable and a finite alphabet of symbols that can be observed at any instant and have dimension o. Considering the model in image A.3, $s = 3$ and $o = 4$.

The probability at any step of jumping from one state to another, or remaining in the same state is represented by the $[s \times s]$ transition matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

where each a_{ij} correspond to the probability of transition from state i to state j.

Then, the probability distributions of emitting a certain symbol while being in a specific state are given by the $[o \times s]$ matrix

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

that for every b_{iz} indicates the probability that the state i have to emit the z symbol.

Finally, the initial state follows the $[s \times 1]$ probabilities

$$\pi = \begin{bmatrix} P(1) \\ P(2) \\ P(3) \end{bmatrix}.$$

A.2.1 Viterbi algorithm

The Viterbi algorithm is a dynamic programming technique used to decode a sequence of hidden states in an HMM by looking at the observation emitted. The algorithm computes recursively

- $V_t(j)$: which is the probability of observing the state j at the instant t;
- $\text{Path}_j(t)$: that is, for every state j, the best path from the beginning to the instant $t-1$ followed by state j.

The final output will be the most probable path (Best Path) and the probability of this prediction (P).

Initialization

The initial probabilities are computed by exploiting the initial probabilities vector π

$$V_1(j) = \pi_j b_{jo(1)} \tag{A.29}$$
$$\text{Path}_j(1) = [j] \tag{A.30}$$

with $j \in S$ (set of states).

Recursion

The recursive step consists of computing all the next steps by maximizing the probabilities

$$V_t(j) = max_i \left(V_{t-1}(i) a_{ij} b_{jo(t)} \right) \tag{A.31}$$
$$\text{Path}_j(t) = [\arg\max_i \left(V_{t-1}(i) a_{ij} \right), j] \tag{A.32}$$

Termination

Finally, the maximum probability from the last three $V_T(j)$ is taken as the output one P, and the corresponding path is chosen as best.

$$P = \max_j V_T(j) \tag{A.33}$$
$$\text{Best Path} = \arg\max_j V_T(j) \tag{A.34}$$

Bibliography

[1] Active mobility: walking and cycling. URL: https://transport.ec.europa.eu/transport-themes/urban-transport/active-mobility-walking-and-cycling_en.

[2] Most centro nazionale per la mobilità sostenibile. URL: https://www.centronazionalemost.it/most.html.

[3] Mouhacine Benosman. Model-based vs data-driven adaptive control: an overview. *International Journal of Adaptive Control and Signal Processing*, 32(5):753–776, 2018.

[4] Debangshu Sadhukhan, Carl Moore, and Emmanuel Collins. Terrain estimation using internal sensors. In *Proc. of the IASTED Int. Conf. on Robotics and Applications*, 2004.

[5] Christian Weiss, Holger Frohlich, and Andreas Zell. Vibration-based terrain classification using support vector machines. In *2006 IEEE/RSJ international conference on intelligent robots and systems*, pages 4429–4434. IEEE, 2006.

[6] Philippe Giguere and Gregory Dudek. A simple tactile probe for surface identification by mobile robots. *IEEE Transactions on Robotics*, 27(3):534–544, 2011.

[7] Yongqiang He, Jun Zhou, Jingwei Sun, Hongbo Jia, Zian Liang, and Emmanuel Awuah. An adaptive control system for path tracking of crawler combine harvester based on paddy ground conditions identification. *Computers and Electronics in Agriculture*, 210:107948, 2023.

[8] Chengchao Bai, Jifeng Guo, and Hongxing Zheng. Three-dimensional vibration-based terrain classification for mobile robots. *IEEE Access*, 7:63485–63492, 2019.

[9] Christopher A Brooks and Karl Iagnemma. Vibration-based terrain classification for planetary exploration rovers. *IEEE Transactions on Robotics*, 21(6):1185–1191, 2005.

[10] David Tick, Tauhidur Rahman, Carlos Busso, and Nicholas Gans. Indoor robotic terrain classification via angular velocity based hierarchical classifier selection. In *2012 IEEE international conference on robotics and automation*, pages 3594–3600. IEEE, 2012.

[11] Jia Xue, Hang Zhang, and Kristin Dana. Deep texture manifold for ground terrain recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2018.

[12] Sebastian Otte, Christian Weiss, Tobias Scherer, and Andreas Zell. Recurrent neural networks for fast and robust vibration-based ground classification on mobile robots. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5603–5608. IEEE, 2016.

[13] Mingliang Mei, Ji Chang, Yuling Li, Zerui Li, Xiaochuan Li, and Wenjun Lv. Comparative study of different methods in vibration-based terrain classification for wheeled robots with shock absorbers. *Sensors*, 19(5):1137, 2019.

[14] Ayushi Chahal and Preeti Gulia. Machine learning and deep learning. *International Journal of Innovative Technology and Exploring Engineering*, 8(12):4910–4914, 2019.

[15] Edmond M Dupont, Carl A Moore, Emmanuel G Collins, and Eric Coyle. Frequency response method for terrain classification in autonomous ground vehicles. *Autonomous Robots*, 24:337–347, 2008.

[16] Mingxia Liu, Daoqiang Zhang, Songcan Chen, and Hui Xue. Joint binary classifier learning for ecoc-based multi-class classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2335–2341, 2015.

[17] Christian Weiss, Hashem Tamimi, and Andreas Zell. A combination of vision- and vibration-based terrain classification. In *2008 IEEE/RSJ international conference on intelligent robots and systems*, pages 2204–2209. IEEE, 2008.

[18] Vojislav Kecman, T-M Huang, and Michael Vogt. Iterative single data algorithm for training kernel machines from huge data sets: Theory and performance. *Support vector machines: Theory and Applications*, pages 255–274, 2005.

[19] Rong-En Fan, Pai-Hsuen Chen, Chih-Jen Lin, and Thorsten Joachims. Working set selection using second order information for training support vector machines. *Journal of machine learning research*, 6(12), 2005.

Ringraziamenti

Con la stesura di questa tesi di Laurea si conclude il mio percorso di studi in Ingegneria Informatica. È stato un privilegio per me poter dedicare questi cinque anni allo studio degli argomenti che più mi appassionano, acquisendo conoscenze e capacità che mi accompagneranno per il resto della vita. Il raggiungimento di questo traguardo non è però stato privo di sfide e momenti difficili, per questo motivo sono grato di essere stato circondato da persone fantastiche che mi hanno supportato e incoraggiato nel dare il meglio di me.

Un ringraziamento va a mia mamma, oltre ad avermi dato questa stupenda opportunità sei sempre riuscita a capirmi e a confortarmi nel momento del bisogno, anche quando io non la rendevo una cosa facile.
Allo stesso modo voglio ringraziare mio papà. Sei stato una grandissima fonte di ispirazione e spero nel futuro di avere sempre la curiosità, l'ingegno e la forza di volontà che ti contraddistinguono. Grazie per avermi dato la possibilità di seguire questo percorso.

Grazie ai miei fratelli Andrea e Nicola, nonostante tutti e tre siamo poco capaci di dimostrare a parole il nostro affetto, ogni volta che ne avete avuto l'occasione mi avete dimostrato quanto in realtà siamo legati e pronti ad aiutarci quando uno di noi è in difficoltà.

Un ringraziamento speciale va a Elga per essere stata presente ogni giorno, anche quando eravamo fisicamente distante, e per essere la persona con cui più facilmente riesco ad aprirmi.

Durante questi anni è stato fondamentale avere degli amici capaci di farmi ridere e che hanno reso questo percorso più facile e leggero.
Grazie Lorenzo e Valentina per le giornate trascorse in università e per avermi accompagnato in infiniti progetti. Non dimenticherò mai il team OSLO.
Non posso non ringraziare Filippo e Nicola per le mille avventure trascorse insieme e per il rapporto unico che si è creato tra di noi.
Ringrazio Chiara e Marta per essermi amiche da una vita, anche se è diventato più difficile vederci avrete sempre un posto speciale nel mio cuore.
Un ringraziamento va poi a Marta per essere fin da subito diventata molto più che semplicemente la compagna di un mio amico.

Infine un pensiero speciale va ai miei nonni Cristino e Lina, Dante e Gianna. Anche se non siete potuti essere con me durante questo percorso, in qualche modo ho sempre sentito la vostra presenza.