SAPIENZA
UNIVERSITÀ DI ROMA

Design and proof-of-concept of the Integration of an Electric Scooter with a Wheelchair: from the Mechanical Design to the Firmware Upgrade

Faculty of Information Engineering, Informatics and Statistics
Master Degree Program in Engineering in Computer Science

Candidate
Luca Tomei
ID number 1759275

Thesis Advisor
Prof. Andrea Vitaletti

Academic Year 2020/2021

Design and proof-of-concept of the Integration of an Electric Scooter with a Wheelchair: from the Mechanical Design to the Firmware Upgrade
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Website: lucatomei.github.io
Author's email: tomei.1759275@studenti.uniroma1.it

Acknowledgments

First, I would like to express my sincere gratitude to my thesis advisor Prof. ***Andrea Vitaletti*** *for his continuous support of my thesis and related research, for his patience, motivation and knowledge.*
In addition to my thesis advisor, I would like to thank Mr. Massimo from ***a.e.o srl*** *for providing us with a folding wheelchair on free loan. Without their support, it would not have been possible to conduct this research.*
I also want to thank my family for their unwavering support over the years in my life and in this thesis.
Special thanks to all my friends who have always been close to me, especially Danilo Marzilli.
Finally, I would like to thank all the Computer Engineering department of Sapienza University of Rome. From what I have experienced, the department brings together a large number of brilliant and passionate students and professors, from whom I have been able to learn pride in dedication and discipline, love for the subjects, and lasting satisfaction in pursuing results. My years in this building have forged me into an engineer, with a keen attention to detail and a deep-seated thirst for knowledge.

Abstract

The growth of cities has led to more cars on the roads, so cars have a great demand not only on infrastructure but also on the environment. This is why it is critical to encourage the use of public transport systems. Public transport systems present a good deal of the problems related to access and convenience for users. At the same time, e-scooter services have been a new phenomenon in micro-mobility and are touted by experts to solve gaps in the way transport systems work in cities. They are also considered a sustainable alternative to cars as they help the realization of an eco-sustainable society. However, the sharing and purchase of electric scooters is limited to users who are able-bodied, thus without mobility problems in the lower limbs.

This study examines how to make it possible for non-abled people to use an e-scooter by making minor software modifications on their vehicle in addition to a docking system for the wheelchair of the end user, a disabled person. This is done by focusing the study on coding and decoding the firmware of an electric scooter and modifying some parameters to support the additional weight of the user sitting in the wheelchair.

It is also necessary to note that the solution reached in this thesis work is substantially economic since it requires, as mentioned above, only a modification on the software side and the use of a metal support that allows the coupling between the electric scooter and the wheelchair. This metal support, having been created using steel as a material, has a cost of about € 50, but in future developments of this project will proceed to engineer this component, for example, aiming to obtain a lower cost rather than a lower weight through the use of composite materials or plastic derivation.

Contents

Introduction and Goals

In recent years, there has been a rapid explosion in the spread of electric scooters, especially in cities and urban districts, due to the fact that they bring benefits on several fronts: they allow sustainable mobility and at the same time permit you to easily reach the various areas of the city or points of interest.
Moreover, their widespread diffusion has been influenced not only by the possibility of purchasing electric scooter at a price very similar to that of a bicycle, and therefore accessible to as many people as possible, but also by the presence of companies that allow the rental of electric scooters, paying according to the time spent or the distance covered, in a similar manner to what happens with other existing services, e.g. cabs.
The diffusion of electric scooters is and has been undoubtedly influenced by the fact that it is possible to use them without the need for a license, without the payment of insurance and taxes of various kinds, and also by the fact that younger people have different tastes than in the past, that is, they are more inclined to develop more environmentally oriented habits and to use subscription or pay-per-use services.
E-scooters can circulate on the street as if they were bicycles and a further advantage is their portability, since once the ride is over they can be folded and transported anywhere (e.g., train, office, schools) due to their negligible weight. Other characteristics favorable to the use of electric scooters undoubtedly reside in their constructive nature, that is, they have a metal alloy structure which makes them resistant to weather, corrosive phenomena and the small problems of everyday life. In addition to this, they have only a battery and an electric motor compared to their entirely mechanical counterpart and put into motion by man.
Obviously, like all things, we cannot fail to discuss their cons, in fact, on the one hand there is currently no legislation in this regard therefore it is allowed to use the vehicle without a helmet or safety accessories and on the other hand it is necessary to increase road safety to ensure the necessary spaces, such as bike paths, and adequate road signs.
So far the use of these means is only possible for people who do not have a severe disability. So with this thesis work is our intention to extend this possibility to people with disabilities in particular going to focus on the mechanical integration between the electric scooter and the wheelchair and the adaptation of the control software of the e-scooter to a different type of load than the common use.

Background

On the basis of what has been said above, the idea of this thesis work was to make the electric scooter usable not only by people with normal mobility but also by people with disabilities or difficulties in movement, so that people from the same family could use the same tool. This choice has brought as a constraint of the project the desire and the need not to make any changes to either the wheelchair or the scooter, making it as easy as possible to couple the two parts and the consequent use by disabled people. Consequently, this project represents a way to approach the needs of both groups of people and thus allow the widest possible public to take advantage of electric mobility and micro-mobility tools and means. Based on the statistics made over the years on the diffusion of disability and electric scooters we will show some data on these aspects to understand how this solution can be applied to reality and how it can improve the life of people with disabilities in order to make it as normal as possible. In the following we will explain the concept of smart mobility and then we will see in detail the spread of scooters in Italy, Europe and the world trying to answer some questions that may arise when approaching the world of micro-mobility.

Smart Mobility

Smart mobility is a new concept found in the transportation and land use literature. This concept is derived from the conventional mobility planning paradigm that mainly aims to improve mobility through the predominant development of road infrastructure. Overall, it should be said that it is based on the development and application of technology for optimizing urban infrastructure, while also taking into account the needs of society. Smart mobility is an important dimension of Smart City concepts and plays a key role in the development of contemporary cities. According to some studies, the main difference between smart mobility and other paradigms is the accessibility of citizens to information, which subsequently has positive implications on reducing CO2 emissions or saving time.

The Shared Mobility Concept

As technology has evolved, shared mobility has begun to play an increasingly important role in travel in urban or rural areas, improving access to work, education, or other services. In addition, some studies have presented the ability of shared mobility to help the environment with better resource allocation, reduced greenhouse gas emissions, increased accessibility, or long-term sustainable behaviors. That said, owning a private car is no longer perceived by the vast majority of young people as an asset; the preference is to pay for and use a shared vehicle. Despite a little publicized beginning at first, the economic, environmental or social benefits brought by shared mobility have made this topic prominent. According to a research study designed to measure smartphone usage in the United States, 74% of respondents said they used their smartphones for location services or to get directions.
The attraction of using smart or shared mobility transportation by citizens is due to the increased availability of using shared data or cloud technologies in order

to meet their mobility needs. It has been observed that from the perspective of sharing operators, the main consumers are people aged 25-35 years old and without children. Moreover, the marketing strategy adopted by these operators aims to attract young people to the sharing habit.
Despite the advantages of shared mobility presented above, obviously, there are some drawbacks perceived by users or potential users when they decide to adopt a service in this category. Shared mobility is not a service in itself but is a concept, consequently it encompasses a wide range of services which in turn each presents "weaknesses" or "barriers" subjectively perceived by each individual.
In fact, extreme weather conditions such as rain or snowfall are negatively correlated with the use of bicycles or electric scooters. An additional factor influencing the use or abandonment of shared mobility services is the level of pollution. The risks of long-term exposure to pollution from car use are generally known, the most affected groups are paradoxically people who use active transportation. This means higher doses of inhaled pollution among bicyclists and consequently a new barrier to the decision to adopt this clean mode of transportation. As a concluding observation of this factor is that the degree of pollution may be insensitive for most people, but vulnerable people (e.g. asthmatics) may be significantly discouraged from using a sharing service again.
Although the concept of shared mobility is new, there are many discussions related to safety and lack of clear guiding policies. For example, bike share services must meet certain standards related to passenger safety such as minimum fork and frame sizes. Evidence to support this can be found in both scientific articles and newspapers: for example, serious accidents have occurred in major European capitals since their introduction. Paris and Barcelona have witnessed the death of several elderly people hit by scooters, while in Brussels an e-scooter user died after falling off the scooter.
According to the evidence presented, it is outlined that the weather, the level of pollution perceived by users, the attitude towards the personal car, the quality of infrastructure for non-motorized travel and the safety of travel are the main factors recognized by citizens when it comes to their decision to adopt shared mobility in general.

The Evolution of Electric Scooter

Electric scooters are becoming an everyday sight on the streets of major cities and could become part of the future of transportation. The following section will present how we arrived at today's electric scooters and take a closer look at the emergence of rental companies in this industry. Several historical developments have contributed to the creation of today's electric scooters, chief among them being a history and evolution that comes from the desire to motorize vehicles primarily through the use of the electric motor. As early as the late 19th century Ogden Bolton Jr. had a patent for an electric motorcycle, interestingly a year before the first gas-powered motorcycle was invented in 1894. The inventor installed a 6-pole DC motor in the rear wheel hub of the bike, while the battery was placed under the horizontal tube of the frame.

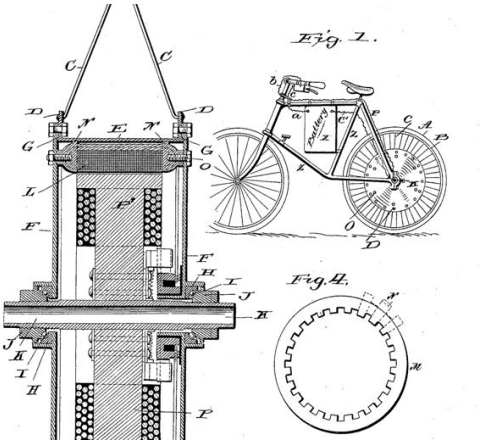


Figure 0.1. Ogden Bolton Jr.'s electric motorcycle patent from 1895

The second major factor in the evolution of electric scooters began with the invention of the kickscooter in Germany in 1817, which came to the United States starting in the 1920s, built by children relying on their imagination. The first scooters were handmade by attaching a handle to a wooden board, which in most cases were rolled on 7.5-10 centimeter wheels with a steel bearing in the center. These scooters could only be ridden by leaning over. The figure below shows an example of these early e-scooters.

Figure 0.2. Early kick scooter with a pair of roller skates

Shared Electric Scooter

Although shared e-scooters are an emerging concept, almost anyone who has had the opportunity to travel in a medium to large urban agglomeration has noticed the presence of this form of micro-mobility. The potential of dockless e-scooters allows for an equitable transportation network for all by creating opportunities to reach underserved areas. However, its rapid expansion has led to unanswered questions that consequently result in further problems expressed by local authorities in their attempt to regulate this new form of mobility (data and privacy, pedestrian safety, speed regulations, parking dilemmas, conflicts with car or bike users, ...).
The first sharing system was introduced in Amsterdam under the name Witte Fietsen (White Bikes) back in 1965. The concept was very simple: bikes were painted pure white to be recognizable and left on the street to be used freely by people. It's not hard to imagine that theft and vandalism quickly caused the system to be dismantled, as it completely lacked counter security measures.
The introduction of physical stations, where users could park and pick up their bikes, along with the implementation of electronic payment systems and bike location tracking technologies, enabled the advent of so-called "docked bike-sharing" (BS) and thus the advent of a third generation of bike-sharing.
The third generation, however, still presented problems with infrastructure set-up and management. On the one hand, the presence of substantial investment required upfront called into question the financial viability of these systems, and on the other, there were structural inefficiencies in vehicle distribution, leading to stations that were either too full or too empty.
Finally, the learning process developed with the previous solutions helped, in 2014, a group of Peking University graduates establish OFO (the world's first dockless bike-sharing system) with the goal of meeting on-campus transportation needs.

In 2017, Bird and Lime launched dockless electric scooters on the sharing market in Santa Monica, USA. In the same year, both Bird and Lime expanded their services to more than 100 cities and reached a valuation of 2 billion in 2018. Subsequently there was the introduction of sharing services from Lyft and Uber, the largest ride-sharing companies in the US and the global scooter market is estimated to be valued at 300 billion to 500 billion by 2030.
The benefits of using such electric vehicles are many, including reducing the use of motor vehicles and the resulting decrease in gas emissions, accessibility, and transforming a city's cycling culture. Accessibility is a very important aspect as the flexibility of driverless sharing, has arguably been the feature most appreciated by users, and the one that has skyrocketed this technology.
With the increased coverage of the system, many commuters have been able to address the so-called "last mile" problem. This term refers to the distance that people must travel from a public transportation hub or station to a final destination, such as home or office. In this sense, combining multiple mobility options, such as "electric scooter + bus/metro + electric scooter" can improve the efficiency of more traditional vehicle-only use. In addition, many cities do not equitably serve their residents with public transit stations, usually cutting off low-income areas from public transit lines. Sharing systems increase equity and accessibility in this regard.

Laws and Regulations

At least 140,000 electric scooters circulate in Italy, including private ones and those of sharing companies, and one in three shared vehicles is an electric scooter.
In Italy, with the law of February 28, 2020 in force since March 1, 2020 (which makes changes to Decree-Law No. 162 of December 20, 2019), the Department of Public Security of the Ministry of the Interior has regulated the circulation on the road of personal mobility devices with predominantly electric propulsion, so, along with segways, hoverboards and single-wheelers, even outside the scope of experimentation, electric scooters.
Based on what has already been established by art.1 c.75 of Law 160/2019, the characteristics to which the electric scooter is compared to a velocipede are established, so not a small moped but a bicycle.
In the same article of Law 160/2019 is addressed the issue related to the circulation that, as a result of the above comparison with velocipedes, should not be subject to special requirements relating to approval, approval, registration, license plate or insurance coverage.
For proper circulation on the road, an electric scooter needs to correspond to certain characteristics:

- Speed: shall be equipped with an electric motor of continuous rated power not to exceed 500W and shall meet a maximum speed of 25 km/h.
- Lights: must have at least one headlight that emits white or yellow light and at least one taillight that emits red light. The front and rear lights must emit a light that is clearly visible from at least 300 feet away.
- Reflective Devices: an electric scooter shall be equipped with at least one white reflective device visible from the front, at least one red reflective device visible from the rear, and at least one yellow or white reflective device visible from both sides.
- Labeling: an electric scooter must be CE marked, a label on the product that designates a set of mandatory practices for all products for which there is a Community directive.
- Weight and Dimensions: an electric scooter must have a maximum weight of 25 pounds, a maximum length of 2 meters, and a width of 0.7 meters measured at the widest point of the vehicle.
- Equipment: must not have a seat or pedals or be attached to a trailer or sidecar.
- Requirements for rain resistance (waterproofing).
- Age: An electric scooter may only be ridden by a person over the age of 15, unless riding is accompanied and supervised by a person of legal age.
- Road Traffic Act: The driver must follow the rules of the Road Traffic Act, which apply to bicycles and bicyclists, including the use of bike lanes, hand signals, and bicycle traffic lights.

- Limits: Drivers must comply with speed limits and the 0.5 blood alcohol limit.
- Liability: Rental companies must obtain liability insurance for their vehicles and the driver must be able to document the insurance to the police.
- Single User: No person other than the driver may be transported on an electric scooter.

Question and Answers

In the following we will explain in detail the spread of scooters in Italy, Europe and the world, trying to answer some questions that may arise when approaching the world of micro-mobility.

Question 1: How many electric scooters are there in circulation?

To date, there are 65.000 shared light vehicles (excluding private ones) offered by 86 micromobility services in Europe and more than 5 million units of electric scooters were sold privately in 2019 in the United States. These are the numbers released by the National Observatory on Sharing Mobility as part of the 4th National Report on Sharing Mobility, which concretely highlight how mobility is changing in our cities. Micro-mobility services are present in a third of the 110 Italian provincial capitals and the best equipped cities in this sense are: Milan, with as many as 14 micromobility sharing services, Rome, with 11 services and Turin, with 7 services.
Scooter sharing is a service that landed in Italy at the end of 2019, but the explosion has occurred over the past few months. Scooter sharing, on par with bikesharing, is the fastest growing micromobility service in the post-lockdown period.
Between December 2019 and September 2020, shared scooters increased from 4.900 to 27.150, a value that is expected to grow in the coming period. Active services increased from 12 to 38 in this same time frame.

There are more than 150,000 electric scooters in about 200 cities in the US and Europe available for sharing. Their standard rates are about 1 dollar to unlock the scooter via the app and 15 cents per minute. So a ride of about 2.5 km costs between 3 and 4 dollars, almost double the cost of bike sharing in the US.

Question 2: How much do electric scooters cost?

There are various brands and models of electric scooters of different quality on the market and their prices generally range from €100 for children's models and €250 for adult models. The cheaper models may not have the best speed, range or general features, but they are still functional.
The price for an electric scooter is determined by various factors, including:

- Specifications such as speed and range;
- Overall quality and material used;
- Extra features should be considered;
- The quality of the battery, in fact the price of batteries varies from 10€ to 25€ each.;
- Software that controls the driving modes.

Question 3: How does sharing work and how much do electric scooters cost to rent?

Electric scooters in Rome in sharing have similar prices and rates. Following the national legislation in force, electric scooters can circulate throughout the municipality on roads with a 50 km/h limit, on suburban roads, on bicycle paths, always maintaining a maximum speed of 25 km/h in the roadway and 6 km/h in pedestrian areas, not on sidewalks.
The way electric scooter sharing works is similar for all electric scooter operators. You download the app for Android and iOS, sign up and enter your credit card information, which is a mandatory step for electric scooter rental in Rome. At this point you will have the coverage map of the service, beyond which you can not park the electric scooter in sharing, and the location of the nearest scooters. Depending on the service you can book them or not. When you are in front of the scooter you will need to scan the barcode or QR code to activate the rental. When you arrive at your destination, you will need to close the rental through the app.
A ride with the shared e-scooter can cost on average from 3 to 5 euros for 6 hours. If a good electric scooter costs between 300 and 400 € it means that in a hundred rides you have already recovered the initial cost.

Question 4: What is the number of people with disabilities?

One out of ten people in the world has a disability. In fact, the United Nations estimates that there are approximately 650 million people with disabilities around the globe. About 80 per cent of them live in developing countries, where one-third of school-age children have a disability - further evidence that development and disability are linked. As the world's population ages, the number of people with disabilities is likely to increase.
In the European Union, the percentage of people with disabilities is estimated by the *European Disability Forum* to be between 10 and 15%, with a total of at least 50 million people.
As far as Italy is concerned, a study by the European Commission using data collected primarily through ISTAT reports that the disabled population in Italy is approximately 2.6 million, or about 4.8% of the total population aged six and over living in families. A figure based on a "narrow" definition of disability (i.e. a total lack of autonomy in one or more aspects of daily life). If we widen the filter, the percentage rises to around 13%, in line with that of other industrialized countries, with a peak of 18.7% for the over-65s.
Of these, 178.000 live in community residences (304 per 100.000 inhabitants), while approximately 153.000 (262 per 100.000) are institutionalized (according to other estimates, 182.000).
As of December 2008, just under 800.000 disabled persons (795.831), including 683.915 males and 111.916 females, had an INAIL pension. Considering the four types of disability, there were 363.152 persons with motor disabilities; 156.873 with psycho-sensory disabilities; 59.584 with cardio-respiratory disabilities and 216.222 with other disabilities.

Question 5: How much can a wheelchair cost?

In general, we can say that the cost of a wheelchair occupies a very wide price range, ranging from € 100 for classic wheelchairs to € 6.000 for motorized ones. Compared to the past, today the market has evolved and has studied in depth the different needs of those who need to use a wheelchair. It is therefore possible to find on the market many different types of wheelchairs, both muscle-propelled and with electric motor. The latter, in particular, could be very interesting to reduce the dependence of many citizens on cars even if they have some limitations, including a maximum autonomy of about 30km in the most severe conditions of use, a maximum speed of 6-12km/h and a too high cost.

Thesis Objectives

As mentioned in the previous passages, the objective of this thesis is to realize the mechanical integration between the electric scooter and the wheelchair and in particular the modification of the control firmware of the scooter is of vital importance. The modification of the firmware is extremely important because it is necessary to adapt the behavior of the electric scooter to the new field of use by tuning parameters and variables of various kinds. For example, it is necessary to reduce energy consumption by deactivating functions that are not useful for driving with a wheelchair and to modify the behavior of the scooter during acceleration and braking in order to make driving more comfortable. In particular, as seen in the previous points, the market for electric scooters is expanding and in the future will increasingly pervade the streets of cities and this combined with the significant number of people with motor difficulties can allow them to take advantage of modern means and environmentally sustainable mobility.
Since the electric scooter can be purchased by families in which there are both able-bodied people and people with impaired mobility, the objective of this thesis project is to propose a solution to be used, without modifying either the scooter or the wheelchair, as a tool for the movement of the two means of transportation. In this way, the vehicle can be used both by disabled people, possible by dragging the wheelchair with the motor of the electric scooter, and also by able-bodied people, since the user will only have to decouple the two elements and use the electric scooter in its canonical use.
In the next chapters we will illustrate in detail the analysis, the design and the implementation carried out at code level to allow the modification of these parameters.

Thesis Structure

In the following chapters, various aspects will be analyzed not only related to the project realized in this thesis work and therefore to the phases of design, implementation and testing, but also regarding the structure and components that characterize the scooter and the wheelchair in order to make understand to the widest possible audience the potential of this project in all its various aspects. Specifically, in Chapter 1 we will analyze the various components that are part of the electric scooter illustrating the functionality expressed by these components. Chapter 2 will show the assembly modalities with relative advantages and disadvantages of each of them. Chapter 3 will deal with the interfacing between the electric scooter and other devices, such as smartphones and computers. In Chapter 4, we will analyze the software component of the project, specifically we will show the phases of analysis, design, implementation and testing of custom firmware created to replace the default one and allow the use by people with mobility impairments. In Chapter 5, we will show the experimental results obtained with the custom firmware and their comparison with the results obtained with the stock firmware. Finally, in the last chapter will be presented the conclusions and final considerations about this thesis project and possible future developments.

Chapter 1

Electric Scooter Components

1.1 Introduction

Electric scooters have recently undergone a considerable boost in terms of diffusion, sustained not only by their low cost and by the possibilities of movement offered by this type of means, but also and above all because of their simplicity of construction, which results in great reliability.
In the following we will briefly see the main components that make up the electric scooters. First of all, of vital importance for the functioning of the scooter is the battery pack which allows the supply of energy and therefore the movement of the scooter through the electric motor, the main component of an electric scooter. In turn, the electric motor will be connected to the rear wheel in which is also installed the module that allows the recovery of energy during braking, that is KERS. On the front wheel we can have a disc brake or a drum brake that allows to assist the braking of the rear wheel. Connected to the battery pack there is an electrical outlet that allows the connection of the power supply to be able to recharge the battery. All these components are connected to the processor that controls and manages the speed and other features related to the movement. In addition, the speed controller will be connected to the throttle, motor and a fuse that allows you to block surges and overcurrents. All these electrical and control parts will be embedded under a tubular structure or under the vehicle's platform on which the user will rest his feet. In order to perform maneuvers, the same user will use a steering wheel on which will be present, in addition to the accelerator knob, a small display that allows to change the parameters of the electric scooter and its driving modes.

The diagram in the figure below shows in an exemplified manner the main components of an e-scooter with particular emphasis on the electrical components as well as the motor.

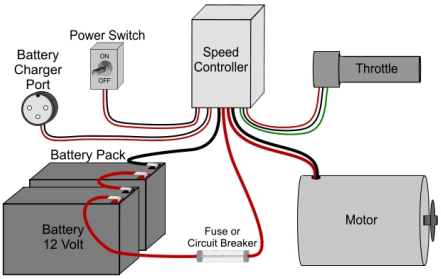


Figure 1.1. Electric Scooter Wiring Schematic

As soon as an electric scooter is purchased, it will have firmware already installed that can be updated via Bluetooth connection through the official application. This firmware is compiled in the ARM-7 architecture and runs on the 32-bit Cortex-M3 processor core of the e-scooter. It allows the riding style of the electric scooter to be modified, changes in acceleration (e.g. quadratic or linear), limits the maximum speed and many other important parameters.

1.2 DRV, BMS, BLE

To understand how the electric scooter firmware works, it was necessary to sniff the packets transmitted by its official application as soon as the smartphone and the electric scooter start communicating via Bluetooth. This is useful for understanding the type of information exchanged between the two parties and what is contained within the firmware binary file. From this operation we were able to understand that the e-scooter uses 3 different firmwares, each of them used to allow communication between the various mechanical parts.

- **BLE**: is the acronym for Bluetooth Low Energy, that is the wireless connection system that our electric scooter has. In the vast majority of e-scooters the Bluetooth Low Energy board is located on the handlebars. This type of firmware is responsible for "talking" with the ESC (i.e., Electronic Speed Controller) and also reads all the data from the brake lever and accelerator. It is also responsible for managing the 3 driving modes that most electric vehicles have (eco, drive and sport), as well as controlling the power button and the light.

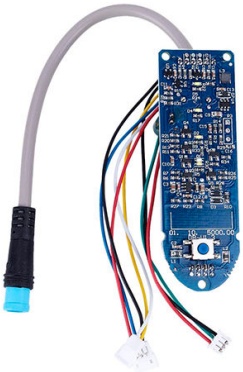


Figure 1.2. Xiaomi M365 BLE hardware components

- **ESC** (or DRV): it is the abbreviation for Electronic Speed Controller, an electronic circuit that controls and regulates the speed of an electric motor.
  The electric scooter has an electronic speed controller which follows the speed reference signal (derived from its manual input lever on the steering wheel) and varies the switching speed of a network of field-effect transistors (FET) [1]. The motor speed is changed by adjusting the switching frequency of the transistors. The motor of the electric scooter is brushless, a motor based on recent technology that offers better performance than motors of previous generations, known as brushed DC motors. Their strengths include greater durability, a better power-to-weight ratio, they are less prone to overheating and also tend to run quietly.
  Most ESC motors are sold with proprietary firmware and in some cases it is possible to modify it either by soldering to connect a programmer or with an alternative open source firmware that can be accessed via Bluetooth.

- **BMS** (or Battery Management System): is an analog and/or digital electronic hardware device integrated with specific software that is added to a battery system. The primary function of a BMS is to meet safety requirements and controls the charging of batteries using variables such as voltage, current, temperature, and the charge of each individual cell, and modifies internal variables such as battery balance rate accordingly to prevent battery damage. If an individual battery discharges, the BMS system will alert the user with its monitoring system. That system can monitor from each battery the current drawn, individual voltage, temperature and calculates the battery charge based on its internal logic system. If the charge of a battery becomes too low, it concentrates the incoming charge from the electrical outlet on that particular battery. While if the maximum charge of a battery becomes too

[1] The FET, or field-effect transistor, is a transistor that uses an electric field to control the flow of current in a semiconductor.

low it will try to increase it. The BMS will cool the batteries if they become too hot and warm them if they become too cold. Objectives related to more efficient use of battery cells and extending battery life are also increasingly being incorporated into BMS design, preventing the battery pack from over-discharging or over-charging.

To find out the version number of each of the 3 firmwares just use the manufacturer's native application or one of the third-party applications available on AppStore and PlayStore, like M365Tools.

1.3 DRV Parameters

The firmware we are interested in modifying is the DRV, in fact we want to bring some precautions to increase the safety of the electric scooter driven in combination with a wheelchair for disabled people and possibly also modulate the performance of the e-scooter trying to reach a good compromise between the power delivered and the battery charge. As mentioned above the firmware allows to manipulate a very high number of parameters. Below we will show some parameters of interest that were used during the project:

- Wheel speed multiplier: Allows to show on the display the correct speed according to the type of tires mounted on the electric scooter. For example Xiaomi M365 mounts 8.5 inch wheels by default, in this case the value is 345. We focus on this parameter because in order to ensure greater driving comfort to a disabled person, it was decided to mount tubeless tires with a larger diameter of 10 inches and therefore not a full tires. In this case the wheel speed multiplier is 315.

- Cruise control: The automatic speed control option has been "borrowed" from cars. Most scooters (but also electric scooters) make use of a similar algorithm to cars for constant speed. Cruise control leverages data received from 3 sensors regarding: engine Rotation-Per-Minute, battery voltage and power used. The electric scooter's control unit uses the signals received from the sensors as input and is able to control the power of the motor. If the sensors detect that the required speed is constant, they keep the motor power constant (i.e., flat road), while if they detect that the speed is decreasing, they increase the motor power until the required speed is reached and kept constant (i.e., uphill road).
  It represents a great advantage especially on long journeys, especially on a straight road, because it can make driving more comfortable and avoid hand fatigue, avoiding having to press the acceleration knob continuously. This function is very convenient if the driver is a normal person, but it could represent a danger in case of a disabled person, as maintaining the speed even when going downhill does not guarantee the user a safe driving. This aspect must be taken into consideration and therefore modulated to avoid causing damage to the person using the electric scooter.

- KERS: Kinetic Energy Recovery System is an electromechanical device that transforms kinetic energy into mechanical or electrical energy during braking, allowing a partial recovery of energy. This system is generally made up of a motor/dynamo, an energy accumulator (electrical or mechanical) and a control system.

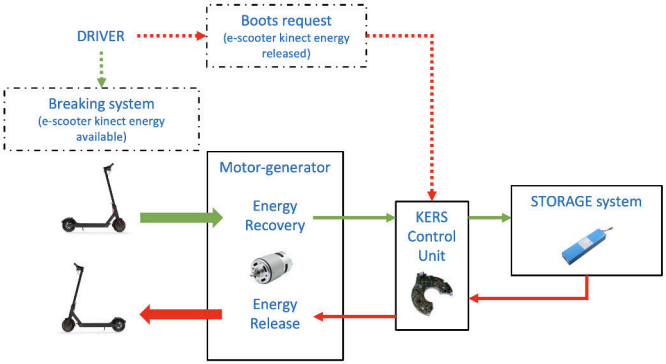


Figure 1.3. KERS functioning

In applications that allow management of electric scooters with stock firmware via Bluetooth, energy recovery is often customizable to 2 or 3 levels, i.e., low, medium, and high based on the amount of energy to be recovered during deceleration and braking. It acts in two phases, when you release the accelerator and when you brake. The first one recovers energy when you release the accelerator, the second one integrates the braking of the main brake with the one produced by the engine brake (for example on Ninebot it brakes electronically at the rear when you activate the drum brake at the front). With the stock firmware the first is deactivable and the second is not.
KERS is an element that not only recovers energy during braking and deceleration but also protects the circuits from the so-called "over voltage" even if many users keep it deactivated without problems.
The KERS is inversely proportional to the "motor power constant" and bound to it. In fact, at low engine power constants, braking will be abrupt and the same applies to when you leave the accelerator; setting a high value for the engine power constant instead, braking will be less abrupt and therefore softer. However, KERS has a negative side, that is, on the stock firmware it acts constantly, both in braking and deceleration and it is not possible to disable it as most of the electric scooters are not equipped with both braking modes: electric brake and drum brake. Precisely for this reason it was decided to make a hardware modification to the e-scooter by mounting an electric brake as an "accessory" that acts as a lever to manually operate the KERS. This modification was made because the sum of the weight obtained by the wheelchair and the electric scooter is considerable and braking with only the

drum brake would increase the braking distance, thus decreasing driving safety.

Figure 1.4. Additional brake lever

In this way it is possible firmware side to disable the automatic KERS and enable it in manual mode to prevent abrupt driver braking.

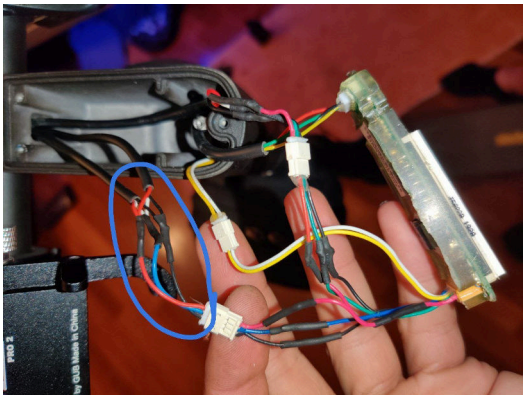


Figure 1.5. Brake lever connections

- Motor Power Constant: Motor Power Constant and KERS parameters are two elements related to each other and necessary when it comes to use custom firmware to modify the power or speed of electric scooters. Speaking in purely theoretical terms, Motor Power Constant represents the ability of the motor to convert the power of electricity into mechanical power. The motor constant is given by:

$$K_m = \frac{T}{\sqrt{P}} \tag{1.1}$$

  where:

  - K_m = constant of the motor $(Nm/\sqrt{Watt})$
  - T = torque (Nm)
  - P = resistive power losses (also known as $I^2 \times R$ losses) (W)

  The Motor Power Constant is a value that increases the power output inversely proportional to the number set, i.e., the lower the number, the higher the power.
  This parameter will have to be decreased to guarantee a greater thrust to the electric scooter when towing the wheelchair.
  Increasing this value will reduce the acceleration and power of the engine, obtaining much more autonomy, while decreasing it you will get an increase in thrust and power, but paying for autonomy. This parameter can assume values in a range between 25000 and 50000.
  After various performance and stability tests it was decided to apply a lower limit of 45000, a value below which it is necessary not to go down.

In the following we will analyze further parameters and the optimal values identified for them:

- Motor Start Speed: Electric scooters need a minimum push to start moving. Through this parameter it is possible to decrease the activation speed of the vehicle. By default this value is set to 5 km/h. This starting speed has been deactivated to allow starting from standstill and to allow towing the wheelchair.

- Current Raising: this value determines the speed with which the control unit passes from one current level to another (acceleration). For the standard 36v battery, a value of this parameter between 1200-1500 mA/step is recommended for rapid acceleration. If in the future you decide to upgrade the battery to 48V, this value can be set to 2500-3000 mA/step for an even more aggressive acceleration.

- DPC Curve: This is one of the most important parameters to modify when the electric scooter is driven by a person sitting in a wheelchair. Stock e-scooters use an acceleration algorithm called "Speed Based Throttle" that makes each position on our throttle correspond to an expected speed. For example a 20% rotation of the accelerator will bring the scooter to 20% of its maximum speed, 50% to 50% until you reach 100% where the scooter will take you to the

maximum possible speed. In practice the electric scooter will force the motor to run at high power until it reaches its intended speed (e.g., 20 km/h), then stop, and then again at the power needed to maintain a constant speed. This process is perceived by the driver as jerky movements, which results in a not perfectly smooth ride. Full power will be delivered only when necessary, e.g., a standing start or a climb.
With the speed-based algorithm, the acceleration curve is linear and proportional, as shown in figure 1.6.

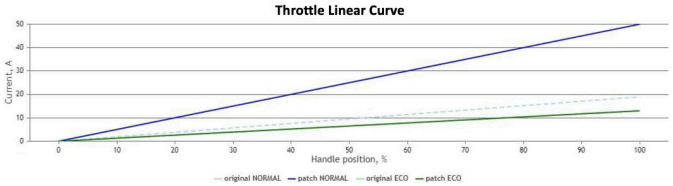


Figure 1.6. Throttle Linear Curve

"Current Based Throttle Algorithms" is a type of algorithm according to which a given rotation of the accelerator corresponds to a total power output in terms of Watts and not to a speed as in the standard firmware of the electric scooter. For example, a half-turned throttle will correspond to a delivery of half the total power of the e-scooter in terms of Watts and keeps it constant. This allows for a much smoother ride because it represents the same principle as any heat engine we have ever dealt with. For example the car, to a total amount of pressure administered to the pedal corresponds an amount of power delivered and not a predefined speed. With 100% throttle with a current based protocol the electric scooter will give the motor maximum power up to the maximum rotation speed then limited by the battery voltage.
With the current-based algorithm, the output power works in logarithmic scale, so the power will be sweet in the low end and then go up on the final. The acceleration curve is shown in the figure 1.7.

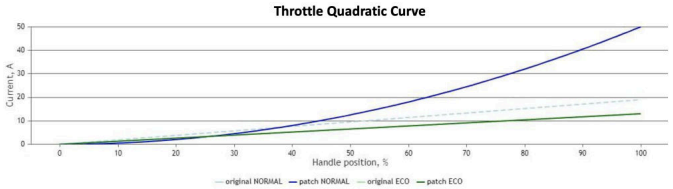


Figure 1.7. Throttle Quadratic Curve

- Max Speed: The title is already quite clear, this item allows you to set the maximum speed in a given mode (Eco, Drive and Sport). This value will have to be decreased to restrict the maximum speed of the electric scooter driven

by a disabled person.

- Version Spoofing: This parameter allows you to trick the official app into believing it has an already updated version, so the user will be able to use the original app that won't offer the official update that would inevitably make the user lose the changes made.

1.4 Electric Scooter Communication Protocol

Before analyzing the components that allow you to interface the electric scooter with devices such as computers or smartphones, it is necessary to know the communication protocols used by the main manufacturers of electric scooters, namely Ninebot and Xiaomi.

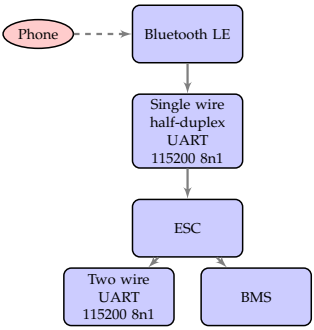


Figure 1.8. Xiaomi/Ninebot Protocol

The protocol illustrated in the figure 1.8 defines the general format with which the Ninebot and Xiaomi electronic control system realizes the multi-node communication through physical serial ports [2] and Bluetooth serial ports. To use the serial port communication it is necessary to use a baud rate [3] of 115200, 8 byte words, without verification and with 1 stop bit. Information retrieval, control and parameter modification can be accomplished through a read-write operation on the "memory control table" which is stored in the controller. The basic storage unit of the memory control table is 2 bytes of signed (short) integer data.
All data longer than 1 byte is subject to the low-priority transmission and storage form.
Xiaomi and Ninebot use the same communication protocol but they exchange packets of different values and size. More precisely, in the instruction packet that reads

[2] A serial port is a communication interface that allows information (incoming or outgoing) to be exchanged sequentially, one bit at a time. A parallel port, on the other hand, is an interface that communicates several bits simultaneously in parallel.
[3] The baud rate in the context of digital transmission indicates the amount of change per unit time of the signal on a transmission channel using modulation or line coding.

and writes the memory control table [4] and in the read response packet, data index indicates the offset address of the data accessed in the table; in the response packet (command word 0x05) with the response write operation, the data index 0 indicates a successful write. Other values are defined as follows:

- 0x01: No write permission for the write-in address.
- 0x02: The control table is under operation, and no write-in is allowed.
- 0x03: Write-in data is out of scope.
- 0x04: Write-in data is in wrong form.

During the firmware download, data index indicates the number of the data packet under continuous data download. When the firmware downloads related instruction response packets, data index 0 indicates download successful. Other values are defined as follows:

- 0x01: Firmware is over-sized
- 0x02: Erase Flash failed
- 0x03: Write Flash failed
- 0x04: Scooter is unlocked or firmware can be updated
- 0x05: Data index error
- 0x06: IAP is busy (such as writing in Flash)
- 0x07: Data format error (length of the data sent is not an integer multiple of 8)
- 0x08: Data verification failure
- 0x09: Other errors

The error values are very useful in case of firmware flashing as they immediately trace the cause of the problem and we can easily understand if the error generated is caused by a problem in the coding of our firmware or by other factors that do not derive from the firmware itself.

1.4.1 Serial port reads memory control table

If the computer wants to read current temperature of the electric scooter, as the index address of scooter temperature in the memory control table is (0x3E), instruction type shall be "read instruction", namely 0x01 and the data index is 0x3E. As the basic unit of the memory control table is a 16-bit integer data, the read length is 2 bytes. Data segment has only one byte with the value of 2 and the frame length is 1. Source ID is ID of the upper computer, 0x3D. Target ID is main control panel ID of the electric scooter, 0x20. Instruction packet sent by the upper computer is:

5A A5 01 3D 20 01 3E 02 60 FF

Upon receiving the data packet, the electric scooter will return current body temperature value. The main control panel will return data to the upper computer according to the source ID of the packet. Therefore, the target ID is the upper computer ID, 0x3D and source ID is main control panel ID of the electric scooter, 0x20. Instruction type is 0x05 (read response) and data index is 0x3E (body temperature). Date segment is the body temperature, as it is a 16-bit data, the data segment has two bytes with the low order in front. Assume current body temperature is

[4] A memory table is directly linked to lists stored in memory. It automatically manages the advanced mechanisms for handling list boxes.

31.8°C, as the temperature unit is 0.1°C, the value read is 318, namely 0x13E in the hexadecimal system. Therefore, two bytes of the data segment are 36 with the frame length of 2. The response packet returned by the electric scooter is as follows:

5A A5 02 20 3D 04 3E 36 01 27 FF

In practical application, in order to save communication bandwidth, multi-data can be read in one time, such as to read 14-byte electric scooter serial number in one time. Start address of the serial number in the memory table is 0x10, with a size of 14 byte, from 0x10 to 0x16. During reading, data index of the instruction packet is set as 0x10 and the read length (namely data segment) is set as 14, the electric scooter will return continuous 14 byte data segment, namely scooter serial number. Instruction packet sent by the upper computer is:

5A A5 01 3D 20 01 10 0E 82 FF

1.4.2 Serial port writes memory control table

When data is about to be written to the memory control table, instruction type of the instruction packet shall be set as "write instruction", namely 0x02 (write instruction with response) or 0x03 (write instruction without response).

- For "write instruction with response", the lower computer will return a response instruction indicating successful write after receiving the "write instruction".
- For "write instruction without response", the lower computer will not return such instruction, whether receiving the instruction or not.

The first type of write is used for write-in of some important data and the upper computer needs to confirm whether the lower computer receives the data or not, such as write-in of electric scooter speed limit value; the latter is used for the occasion having high requirement on real-time performance but paying less attention to the write-in, such as write-in of remote control target speed value under remote control mode.

For example: write electric scooter speed limit value 10 km/h under the speed limit mode with response. Index of the speed limit value is 0x74 with the unit of O.1km/h. So, the value written is 100, namely 0x0064 in the hexadecimal system. As the basic unit of the memory control table is a 16 bit integer data with the low order in front during writing, the data segment to be written is 0x64 0x00, and the whole instruction packet is:

5A A5 02 3D 20 03 74 64 00 C5 FE

In case of successful writing, the response packet returned by the lower computer is as follows:

5A A5 01 20 3D 05 74 01 27 FF

Like read operation, multi-data can also be written in the write operation, such as writing of Bluetooth pairing code 123456:

5*A* *A*5 06 3*D* 20 03 17 01 02 03 04 05 06 6*D* *F**F*

Chapter 2

Assembly between Wheelchair and Electric Scooter

In this chapter we will show the various ways in which the mechanical coupling between the two parts has been provided, ie between the wheelchair and the electric scooter. Specifically, two main ways of interconnecting the parts have been identified, but each of them has advantages and disadvantages that we will also illustrate through images showing the type of assembly. In both cases, the support that allows the connection between the two parts is the same therefore the use of one mode over another does not affect the components and is at the discretion of the end user. More specifically, a support has been created to be installed at the base of the wheelchair structure near the rear wheels, thus becoming an integral part of the wheelchair structure. A movable component is added to this support which, by means of a knob, allows the electric scooter to be fixed to the aforementioned support.
Specifically, three knobs were used connected to three respective threaded rods and three 3D printed nylon anti-vibration pads. The threaded bars have been cut to size and, through a joint in the part that connects them to the anti-vibration mounts, allows any type of electric scooter to be kept firmly on the support with respect to the longitudinal and lateral movements produced by acceleration, braking and steering phenomena. Furthermore, the knob placed in a horizontal position with respect to the ground allows to raise the position of the front wheels of the wheelchair which, in both assembly modes, would create problems of a kinematic nature. In this sense they are "disabled" in their functionality and therefore the movement of the wheelchair will be subject only to the traction and steering imposed by the electric scooter. In addition, by means of a steel hinge, the mobile part connected to the support allows the electric scooter to enter under the wheelchair without it having to be lifted or moved. In the following, images relating to the various parts of the coupling and the components used will be shown.

Figure 2.1. Bracket mounted on the wheelchair

Furthermore, since electric scooters do not have reverse gear at the level of the electric motor, it is necessary to ensure that this is still done even by those with disabilities. In this sense, the structure we have built does not interfere in any way with the user's ability to go backwards as he would have done with a wheelchair not equipped with the scooter. Specifically, by acting on the rear wheels of the wheelchair and making the steering using the handlebar of the electric scooter, it is possible to reverse without any effort without the e-scooter being an obstacle in

carrying out these actions.

2.1 First Assembly Method

The first assembly method requires that the rear wheel of the electric scooter is located before the "x" structure, that is the one that allows the wheelchair to be folded. The use of this mode requires that the steering wheel of the electric scooter is folded because otherwise it would be too far from the wheelchair seat. In this sense, the use of the folded steering allows the user to enjoy a wide view and at the same time to carry out steering maneuvers easily and without fatigue. On the other hand, however, the turning radius is considerably reduced, ie by about 20% of the steering wheel of the electric scooter, which in fact reduces the steering capacity of the electric scooter. Some images related to this assembly method and its use will be shown below.

Figure 2.2. First assembly method

In light of this problem, a further method of assembly has therefore been envisaged, although this is the one that has the most advantages at the current stage of development.

2.2 Second Assembly Method

The second assembly method foresees that the position of the rear wheel is located before the "x" structure of the wheelchair described above. In this case, unlike the previous driving mode, the handlebar is located much closer to the user so it is not necessary to use it folded and this allows you to take full advantage of the turning radius of the electric scooter. On the other hand, however, this configuration reduces the end user's field of vision since the steering will be located at a higher height than the user's shoulders. This is because the electric scooter is designed to be used in an upright position. Obviously this problem would be solved when using an electric scooter with telescopic steering, therefore able to support different heights for the steering.

Some images related to this assembly method and its use will be shown below.

Figure 2.3. Second assembly method

Chapter 3

E-Scooter mobile/computer interfacing

It is possible to interface the microcontroller of the electric scooter with various electronic devices according to the technology that can be used. In particular we can have interfacing through serial port or through Bluetooth connection (Bluetooth Low Energy).
In the first case the interfacing will be possible through the use of a computer and one of its USB ports that will communicate with the BMS, while in the second case it will only be necessary to use a device with a Bluetooth connection.

3.1 BLE - Bluetooth Low Energy

The purpose of this section is to explain the aforementioned technology in more detail since it was actively used in the thesis project and therefore it is important to understand its theoretical foundations and its practical implementations.
The Internet of Things (IoT [1]) architecture has become popular in recent years because it has brought new solutions to needs already present in today's world through the massive use of new technologies, particularly related to telecommunications (e.g., 4G, LoRa WAN) and modern artificial intelligence techniques.
Bluetooth, in its standard version, is an example of a technology that has several limitations when used in scenarios that transcend the use cases for which it was intended. For example, it does not allow communication between more than one device at a time, has a limited range and few countermeasures to counter interference.
Over the course of its history, new versions have been defined that have, from time to time, improved the capabilities and characteristics of the protocol and filled in its shortcomings. For this reason, Bluetooth has incorporated in a recent version support for the Low Energy standard, which allows its use in applications that need to use Bluetooth as a means of communication without requiring large amounts of energy.

[1] Internet of Things, is a neologism referring to the extension of the Internet to the world of concrete objects and places.

The Bluetooth Low Energy protocol project was started by Wibree, a company that is part of the Nokia group and which has always been interested in the development of telecommunications technologies. The priority objective of the engineers and designers was to define and implement a radio standard that would allow the lowest possible energy consumption and at the same time be simple, making it possible to use it in low cost systems. In other words, these assumptions have proved ideal for the smartphone and Internet of Things market where devices are almost always powered by small batteries and at the same time do not require high transmission capacities. In 2010, Bluetooth Low Energy, also known as Bluetooth Smart, was incorporated into Bluetooth version 4.0 along with the Bluetooth Classic protocol shown in Figure 3.1.

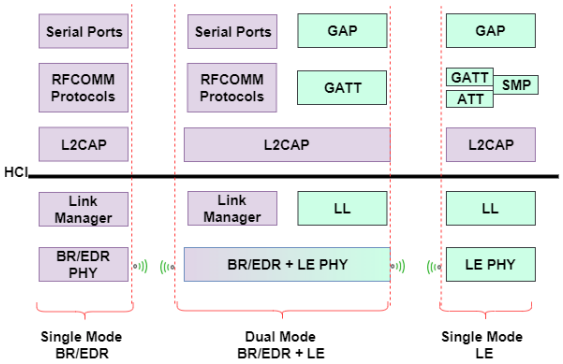


Figure 3.1. The architecture of the Bluetooth stack

However, these two communication protocols should not be confused, because apart from the physical layer (2.4 GHz radio interface in the ISM band [2]), they differ significantly in their design intentions. Bluetooth Classic is a typical protocol that enables fast communication however it requires a significant amount of power. In December 2013, the first major change to the protocol was introduced (Bluetooth version 4.1), followed a year later by further changes in the form of the Bluetooth version 4.2 standard. Bluetooth Low Energy has some limitations due to its main premise which is to ensure a high level of energy efficiency. However, due to this important boundary condition a first consequence is related to the achievable data throughput: the upper limit of the transmission speed is 1 Mb/s, only theoretical value. In fact, in practice, this value is subject to many physical and hardware limitations imposed by system manufacturers. The standard specifies that a single data packet can contain a maximum of 20 bytes and consequently the hardware limitation comes from the packet sending frequency.
Most electric scooters use semiconductors manufactured by Nordic, a company based in Norway. This company specialises in wireless systems-on-chip (SoC) with

[2]The ISM Band is the name given to a set of portions of the electromagnetic spectrum reserved for radio communications applications for industrial, scientific, and medical use.

the aim of producing connectivity devices for the 2.4 GHz ISM band to ensure large energy savings. Typical end-user applications include wireless mobile phone accessories, wireless mice, gamepads and keyboards, wireless medical devices, smart sports equipment, remote control, wireless voice audio applications (such as VOiP) and security.
For Nordic Semiconductor's nRF51 family microcontroller, this limitation amounts to a maximum of 6 packets per connection interval. This is a configurable parameter that determines the time period within which, if the packet is not transmitted, the connection will be considered broken. The connection interval can be between 7.5 ms and 4 s, so if you take the lowest value of this parameter, you get the bandwidth:

$$\text{Bandwidth} = 6 \text{ packets/interval} \cdot \frac{1000ms}{7,5ms} \cdot 20\text{Bytes} = 15960\text{Bytes/s} \approx 128KB/s$$

$$160\text{bit}/1000000 \text{ bit/s} = 1.6 \cdot 10^{-4}s$$

$$7.5ms = 7.5 \cdot 10^{-3} \rightarrow 7.5 \cdot 10^{-}3/1.6 \cdot 10^{-4} = 10 \cdot 4.68 = 47 \text{ packets}$$

As you can see from the equation it is a value that is very different from 1 Mb/s, however, compared to the gain in energy consumption, it is still a very good result.

Another limitation of BLE is the communication distance. Officially Bluetooth Low Energy has a range of 50 m, but it is a theoretical value and therefore difficult to achieve. This value highly dependent on the environment (there must be no obstacles between devices), the transmission power (re-configurable, the lower the range) and the number of other devices nearby (interference between different devices). A large number of Bluetooth Low Energy transmitters affects the occupation of communication channels and thus further reduces the bandwidth of the link. The frequency band (2.402 GHz to 2.480 GHz) used by the protocol is divided into 40 channels, as shown in the figure 3.2.

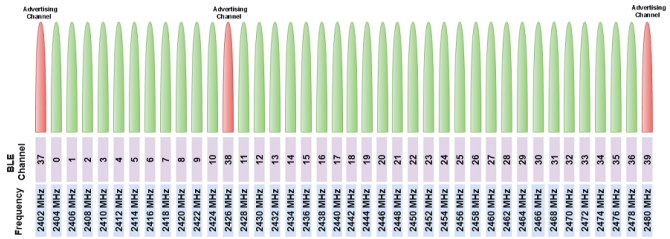


Figure 3.2. Frequency Band Bluetooth Low Energy

3.2 Connection via Bluetooth Low Energy (BLE)

Electric scooters use Bluetooth Low Energy (BLE) to communicate with the user's smartphone. E-scooter achieves this communication through Nordic Semiconductor's nRF51822 chip, which is an ultra-low power 2.4 GHz wireless system on chip 16, as described in Section 3.1. In figure 3.3 we can see the official iOS App for the Ninebot electric scooter that can be used to retrieve information about it and manage its settings:

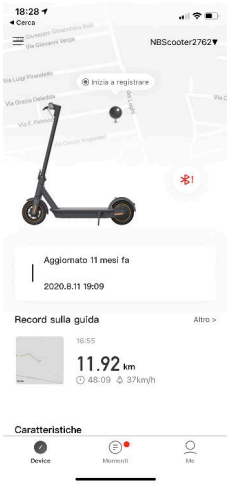


Figure 3.3. Ninebot iOS Application

The electric scooter comes with pre-installed firmware, which can be updated from the manufacturer's app via Bluetooth Low Energy. The firmware runs on the e-scooter's 32-bit Cortex-M3 processor core and is compiled on the ARM-7 17 architecture. It defines many aspects related to the functionality of the electric scooter, including the internal speed limit, defines the power supplied by the battery to the motor, the speed of acceleration and more.

The exchange of data over Bluetooth Low Energy occurs after the pairing process between the connected devices and usually takes less time than a traditional Bluetooth connection. Both the authentication at the moment of the connection request and the encryption of the exchanged data are decided by the developers of the system used. This feature is suitable for situations where it is necessary to support a long-term connection with little data exchange between two devices. Several tools can be used to be able to perform the BLE connection, such as *hcitool*, *gatttool* or *BetterCAP*. Once *BetterCAP* is launched, we can proceed with a Bluetooth Low Energy scan using the `ble.recon` command, which will search for BLE devices and access their advertising services. We can then stop scanning once our device is

found and save its address. In figure 3.4 we can see the results of the scan command with *BetterCAP* tool.

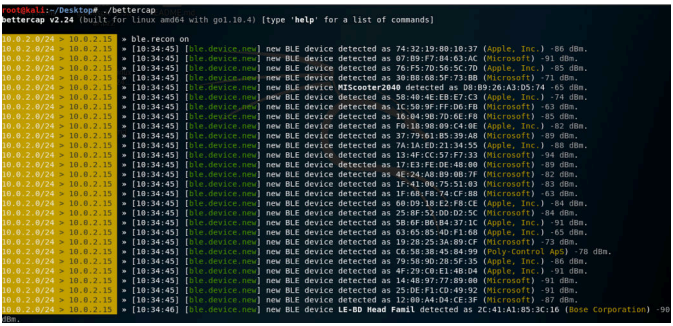


Figure 3.4. Bluetooth scan via Bettercap

At this point we can perform a multitude of actions on the device associated with this address. Bettercap will attempt to connect to the electric scooter first, perform the action, and disconnect immediately. We can enumerate its information using the `ble.enum`. Through the command `ble.enum MAC ADDRESS` we can also update the characteristics of the BLE firmware, values that we don't know yet and that can be found in several ways: by reverse engineering the e-scooter firmware of the mobile application or analyzing (i.e. sniffing) the traffic between the electric scooter and the application. Usually the values that describe the characteristics of the device and that are written on Bluetooth Low Energy are encrypted and authenticated, however the firmware sniffed by the application and the data obtained from the server requests didn't have any kind of encryption.

A cybersecurity company named Zimperium in 2019 publicly released a piece of code that allows them to lock and unlock any Xiaomi electric scooter. They released their exploit in the form of an Android app that allows an electric scooter to be unlocked or locked regardless of the password set by the owner by using the phone's Bluetooth capabilities to interact with the e-scooters to perform nefarious actions. According to Zimperium, *"During our research, we determined that the password is not properly used as part of the authentication process with the electric scooter and that all commands can be executed without the password. The password is only validated on the application side, but the scooter itself does not track the authentication status."*

Through the use of Bettercap it was possible to perform a man-in-the-middle attack using one of the best BLE hacking software, gattacker. This allowed us to monitor traffic between a smartphone running the Ninebot or Xiaomi app and the electric scooter. In this way we were able to activate the lock and unlock commands on the app and took note of the data sent from the phone to discover the values used to initiate these actions in the scooter. This procedure was simply done in an attempt to learn more about the Bluetooth Low Energy process and to get the firmware

values that would then be modified to finalise this project. Using the read command in Bettercap we were able to get more information regarding the Bluetooth Low Energy firmware of the electronic scooter we are analysing. This is shown in the figure 3.5:

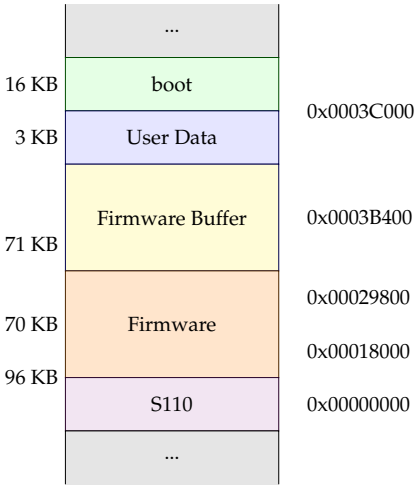


Figure 3.5. Xiaomi M365 Bluetooth Low Energy Stack

3.3 Serial Connection

Serial connection is a much more complex and articulated procedure compared to flashing the firmware by phone, since the latter uses only a device with Bluetooth connection. In order to connect the electric scooter to the PC via serial connection, it will be necessary to disassemble it, remove the BMS controller and once the soldering has been done on it, it will be possible to connect a ST-LINK cable to the computer and flash the firmware.
As shown in figure 3.6, you must solder 3 dupont cables to the positive, negative and ground of the PCB and use the STM Utility programmer of ST-LINK that will allow you to connect to the ESC and import the binary firmware to write directly on the board.

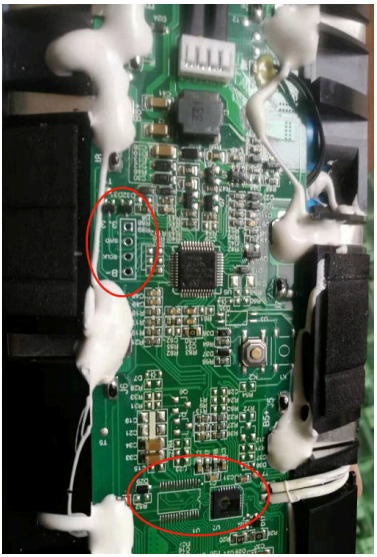


Figure 3.6. Ninebot Max G30 PCB connection pins

This section will not be discussed in detail because the custom firmware generated for this project will be flashed through a simple iOS or Android application and therefore this mode of access to the firmware of the electric scooter, although available, will not be used for the purposes of this project. It was mentioned for the sole purpose of showing the multiplicity of ways in which you can access the firmware of the e-scooter but unless specific reasons is always preferable to use the most convenient and less invasive mode.

Chapter 4

Modify firmware parameters

Electric scooters communicate with the smartphone app using established and prevalent technologies such as the Bluetooth Low Energy. However, the use of such communication channels also opens the door to too many attacks, some of which may be particularly easy and effective on electric scooters.

4.1 Sniffing Packets

To obtain the original firmware of the electric scooter used for this project, a Man in the Middle attack was performed by "sniffing" the communication between the two entities.
Sniffing is defined as a process that refers to the investigation of something hidden to find sensitive information. More specifically, sniffing refers to intercepting traffic or routing traffic to analyse, monitor and capture data traffic. This technique is mainly performed to analyse network usage, solve network problems, monitor sessions for development and testing purposes.
Network packets or TCP/IP packets contain information required for two network interfaces to communicate with each other. It contains fields that are source and destination IP addresses, ports, sequence numbers, and protocol type. All these fields are vital for the proper functioning of the various network layers, especially for the Layer 7 application that uses the received data.
Man-in-the-middle is a cyber threat that allows the cyber attacker to intercept and manipulate Internet traffic that the user believes is private. More specifically, in the case of a man-in-the-middle attack, the attacker literally puts himself in the middle between two entities that are trying to communicate with each other: a client (the victim) and the server or router. In this way, he can not only intercept the messages sent and received, but he can also modify them or pretend to be one of the two parties.

The figure 4.1 graphically illustrates this type of cyber threat.

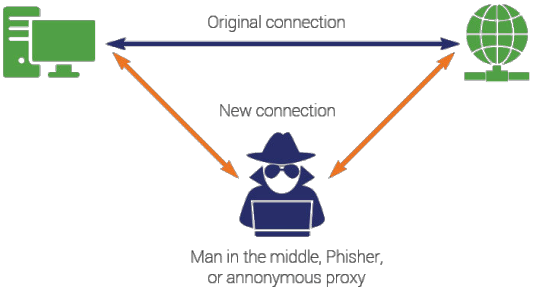


Figure 4.1. Man-in-the-middle attack

Thor HTTP Sniffer/Capture is one of the best network packet analyzers for iOS, it is an application that monitors traffic data to and from a computer network connection. It serves to analyze packets with the purpose of presenting captured packet data in the most detailed way possible by intercepting binary traffic by converting it into readable format, making it easy to identify what kind of traffic is going through the network, how much and how often.
After installing the application, clicking on the single button on the main screen (as shown in Figure 4.2) sets up a DevTunnel session through which Thor will be able to capture packets crossing the network to analyze them later. Now just turn on the electric scooter and start the official application (Ninebot) to allow communication between the two entities.

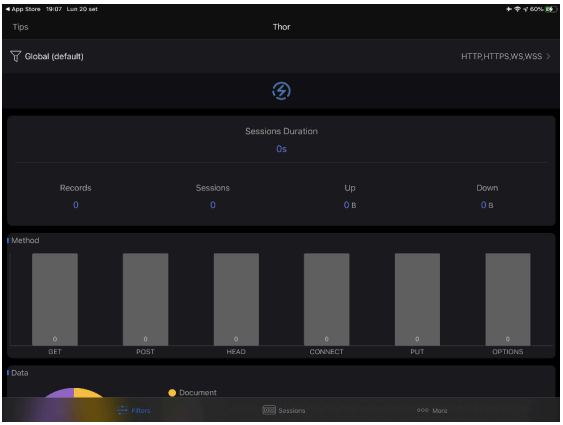


Figure 4.2. Thor HTTP Sniffer/Capture iOS application

4.2 Analyzing sniffed packets

Once the traffic has been monitored, it is enough to close the DevTunnel session and export the archive in *.har* format to carry out a careful analysis on the computer using Wireshark. Through this sniffing tool we were able to reconstruct a scheme that describes the type of communication that the electric scooter makes while communicating with the cloud server of the parent company. In figure 4.3 we can see how the communication works among the various actor involved in it.

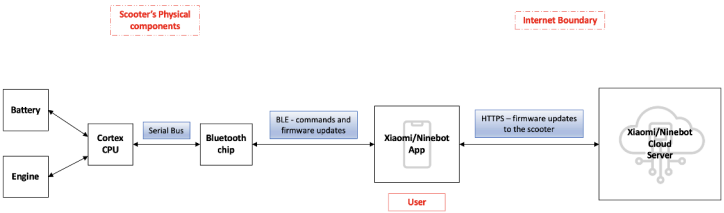


Figure 4.3. Scooter-App Communication Diagram

By analyzing the various HTTP requests and responses we find an end-point that provides through simple GET/POST the firmware of the electric scooter previously connected to the application. To obtain the binary file of the firmware we created a Python program that through three simple functions performs the same operations of the official application saving the result locally. Through this program we were able to obtain information about the date of activation of the device, the km made, the state of the batteries and our personal data including phone number and email.

The figures below show the result obtained from the execution of the Python program:

```
{
   "login_key":"YXBwOkFvaDk5SkxRV2dRaEx0WWc=",
   "resource_url":"http://52.57.72.217/app"
}
```

Figure 4.4. Obtaining Access Informations

```
{
   "code":1,
   "data":[
      {
         "wnumber":"N4GCD2****2762",
         "vehicle_type":"54",
         "auth_uid":"27***754",
         "auth_phone":"**********",
         "auth_email":"luca*******@gmail.com",
         "auth_date":"1597156103",
         "total_mileage":"1.2",
         "ble_name":"NBScooter****",
         "vehicle_name":"Ninebot KickScooter Max G30",
         "...":"..."
      }
   ],
   "desc":"Mission Accomplished",
   "tip":""
}
```


Figure 4.5. Getting User Credentials


```
{
   "code":1,
   "data":{
      "is_test":false,
      "ctrl":{
         "last_version":"355",
         "version_content":"1. Optimize firmware compatibility...",

"bin_content":"0fe35d59357a4e53c8c8d7c399a495b2370e3d2984850e6861d24de9453ad0f
60969059101beee2b3828c2796775583da8300f5174f037c90aac0f4f5ed1649c61ed62fe838b2
2b273677306b0aeb82a6b4df49b9adac61e9b11cc3d5807ed8823962b4c86ec131f7f4ef5a8d0d
7a633fb6047e75aec7060b3fe434b9d302d86d7935a962be7e1b4ca42df3e9e5579494a81b6507
fbe315e3afd7d6976b40658a4b5414a26ad0a4c89e07b477b5448ca3ee633e905d97df2e34b04a
2ec5d33757027af00fab90295177469dfe4fdfb77ffb96ed8d2e31506788a86ac4c49fc7c3f123
afbda2a070ce908a3530f7f75ced2e6d5564bbde4934eadb4bc99744...",
         "verify_code":"67650446",
         "firmware_type":1
      },
      "ble":false,
      "bms":false,
      "forced_status":false,
      "forced_content":"",
      "special_version_status":false
   },
   "desc":"Operazione riuscita.",
   "tip":""
}
```


Figure 4.6. Getting last DRV firmware

Going to analyze the last request to the server we can see that the json key *bin_content* returns in clear the binary file that will be used as source for the parameter modification operation.

4.3 Firmware and Custom Firmware

The term **firmware** derives from the union of the two words "firm", that means stable, and "ware", that means component. This term indicates a particular type of program, normally stored on a small non-volatile memory, that in a certain sense is placed inside a desktop computer, a notebook, a smartphone, a tablet, a modem router, an electric vehicle and in general inside any other type of electronic device. The task of the firmware is therefore to correctly start the device in question while allowing, at the same time, the dialogue between the hardware of the device and its software.
In the case of an electric scooter or any other type of electric vehicle, the firmware or software is updated wirelessly OTA (Over-The-Air) using a telecommunication device, through a gateway inside the vehicle itself, to the ECU (Electronic Control Unit), a small device responsible for controlling some vehicle functions whose presence is increasing in modern vehicles. With over-the-air technology, updating a vehicle becomes much easier for both manufacturers and vehicle owners. In fact, as shown in the figure below, manufacturers store their updates in their cloud server and vehicle owners can choose any time to install the update while connected to a Wi-Fi network. This allows the vehicle to perform the update from the comfort of the owner's home, a significantly simpler, more convenient and less time-consuming process.

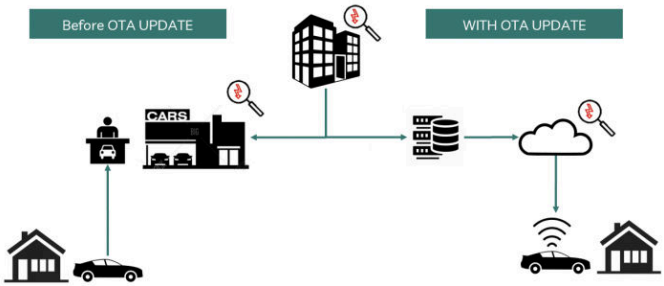


Figure 4.7. Vehicle Update History

The **custom firmware**, also known as aftermarket firmware, is an unofficial version of firmware created to provide new features or unlock hidden features by third parties on devices. The firmware produced by the parent companies of electric scooters have parameters that often make our medium work in a very relaxed way, not using the full power or otherwise making it operate in limits of hyper security. The firmware can also impose speed limits imposed by local law depending on the area where the product comes from. Modifying a firmware means to modify the parameters defined by the parent company. So we can increase the maximum speed and acceleration, as well as several other parameters to adapt it to our needs and driving style. Creating a custom firmware has various pros and cons. As a pro it is clear that you have the possibility to give extra performance such as speed and

torque vehicle but on the other hand this will affect the battery life and hardware, especially parameters are set exaggerated or not appropriate to the driving style. In fact it doesn't make much sense to look for parameters at the limit of electronics but rather safe parameters keeping the same electric scooter a more performing and at the same time reliable and safe means.

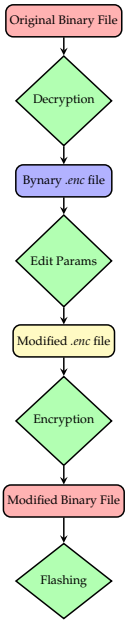
4.4 Electric Scooter Stock Firmware Modification

Before proceeding with binary file editing, it is necessary to analyze what this type of file is and what its use is. In most contexts, files are divided into two categories: text files (can be read by humans) and binary files (can be read by a computer). Text files are stored as a sequence of encoded characters, each comprising 8, 16, or 32 bits of binary. The characters are relatively easy for humans to decode, if you know the character set used. Binary files are a computer-readable form of data storage, and a program is required to interpret that data and display its contents to the user. In fact, a binary file is not in any externally identifiable format so that any program that wanted to could search for certain data at a particular point within it. A program (or hardware processor) must know exactly how the data is arranged inside a binary file in order to use it since it is usually not readable by any text editor.
Usually this type of files have a file name extension of *.bin* and are often used for executable programs because they may contain data that is ready to be used by a program.
A first careful analysis of the binary file obtained by sniffing packets from the original application was performed with a decompiler that transforms this data structure into a readable assembly and then into a python data structure (*bytearray*) that automates through various functions this process. The software used for this task is Ghidra, a tool developed by NSA that contains a set of reverse engineering tools. This software was very useful as we were able to decompile the firmware to identify the parts of the code that contain the internal variables relating to the parameters described above, thus reproducing the modification of some of them through a Python program.
Once obtained the firmware binary file from the last request we must decrypt it and then proceed with the modification of the same. Through the *"EncDec"* class we have implemented functions to encode and decode the Xiaomi and Ninebot firmware binary files. This class extends functions originally implemented by a user in a Russian forum that were used only to encrypt/decrypt Xiaomi firmware. Since the electric scooter in question is from another manufacturer (Ninebot), they have been modified and extended to make them compatible with this type of electric scooter. This class takes as input the binary file "sniffed" by the Ninebot/Xiaomi application and returns as output the decrypted file for subsequent parameter modification. These functions can also be used to encrypt the decrypted file by adding additional parameters for later flashing to the electric scooter. In listing 4.4, we can briefly see the code of the encrypt and decrypt method used in the project.

```
# With this module you can easily encode or decode Xiaomi-Ninebot firmware
    files or make ZIP archives
# It takes as input the binary file sniffed by the Ninebot / Xiaomi application
     and returns the decrypted file for the subsequent modification of the
    parameters.

# We can use the same module to encrypt the previously modified and decrypted
    file

def encrypt(self, data):
  data = self.Binary_Utilities_OBJ.pad(data)
  assert len(data) % 8 == 0, 'data must be 8 byte aligned!'
  res = bytearray()
  for i in range(0, len(data), 8):
    ct = self.Binary_Utilities_OBJ
        .ecb_enc(self.Binary_Utilities_OBJ.xor(self.iv, data[i:i+8]), self.key)
    res += ct
    self.iv = ct
    self.offset += 8
    if (self.offset % 1024) == 0: self.update_key()
  return res

def decrypt(self, data):
  assert len(data) % 8 == 0, 'data must be 8 byte aligned!'
  res = bytearray()
  for i in range(0, len(data), 8):
    ct = data[i:i+8]
    res += self.Binary_Utilities_OBJ
        .xor(self.iv, self.Binary_Utilities_OBJ.ecb_dec(ct, self.key))
    self.iv = ct
    self.offset += 8
    if (self.offset % 1024) == 0: self.update_key()
  return self.Binary_Utilities_OBJ.unpad(res)
```

Listing 4.1. Main methods for encryption/decryption

To proceed with the actual modification of the parameters of interest in the firmware this logical scheme has been followed:



To modify the decrypted firmware three classes have been implemented: *Make_Firmware*, *Firmware_Utilities* and *Binary_Utilities* . The *Make_Firmware* class is the main class and implements several methods that allow to modify single firmware parameters such as maximum speed, KERS, Acceleration, Motor Power Constant, and so on.
For each of these parameters we need to obtain the pattern in the binary structure of the data to modify using the *GetPattern* function. It takes as input the binary file converted into a *bitearray* and performs a search in this data structure returning in output the index to be used to modify the desired value.

```
39 # To find the pattern
40 def GetPattern(self, data, signature, mask=None, start=None, maxit=None):
41   sig_len = len(signature)
42
43   if start is None: start = 0
44   stop = len(data)
45
46   if maxit is not None: stop = start + maxit
47
48   if mask:
```

```
    assert sig_len == len(mask), 'mask must be as long as the signature!'
    for i in range(sig_len):
      if signature[i] is not None:  signature[i] &= mask[i]

  for i in range(start, stop):
    matches = 0
    while signature[matches] is None or signature[matches] == (data[i + matches
    ]
        \ & (mask[matches] if mask else 0xFF)):
      matches += 1
      if matches == sig_len:  return i

  raise SgnException('Pattern not found!')
```

Listing 4.2. Get pattern function

Each function designed to modify the single parameters of a firmware is described in the *make_firmware.py* file and some of the methods that the class implements are shown below.

```
# This type of algorithm replace the linear throttle curve with a quadratic one.

# With the quadratic curve, the power delivery works on a logarithmic scale,
# sweet at the low end and then soaring on the final.
def throttle_alg(self):
    sig = [0xF0, 0xB5, 0x25, 0x4A, 0x00, 0x24, 0xA2, 0xF8, 0xEC, 0x40, 0x24, 0
    x49]
    ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 4
    pre, post = self.data[ofs:ofs + 1], bytearray((0x01, 0x24))
    self.data[ofs:ofs + 2] = post
    return [(ofs, pre, post)]
```

Listing 4.3. Throttle Algorithm function

```
# Motor start speed
def motor_start_speed(self, kmh):
    ret = []
    val = int(kmh * 390)
    val = val - (val % 16)
    assert val.bit_length() <= 12, 'bit length overflow'
    sig = [0x4B, 0x01, None, None, 0x48, 0x00, None, None, None, 0xB6, 0xF5, 0
    xC5, 0x6F, 0x0D, 0xDB]
    ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 9
    pre = self.data[ofs:ofs + 4]
    post = bytes(self.ks.asm('CMP.W R6, #{:n}'.format(val))[0])
    self.data[ofs:ofs + 4] = post
    ret.append((ofs, pre, post))
    ofs += 4

    pre, post = self.data[ofs:ofs + 1], bytearray((0x0D, 0xDB))
    self.data[ofs:ofs + 2] = post
    ret.append((ofs, pre, post))
    return ret
```

Listing 4.4. Motor Start Speed function

```
# Minimum speed below which the kers is activated
def kers_min_speed(self, kmh):
    val = struct.pack('<H', int(kmh * 390))
    sig = [0x25, 0x68, 0x40, 0xF6, 0x24, *[None]*2, 0x42]
    ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 2
    pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 4, val,
    MOVW_T3_IMM)
    return [(ofs, pre, post)]
```

Listing 4.5. KERS minimum speed function

4.5 Flashing new firmware

To facilitate and try to automate the phase of modifying the firmware of the electric scooter a tool has been created that with the help of a graphical interface allows the creation of the modified firmware through the simple click of a button.

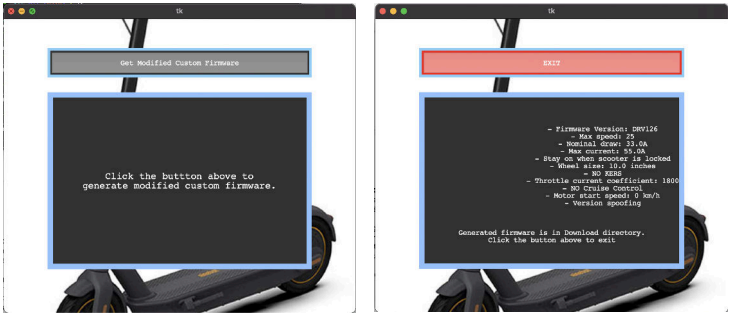


Figure 4.8. Graphical interface of the firmware generator

At the end of the firmware generation wizard, clicking the "EXIT" button will save the firmware to the download folder in .zipper format. This archive will contain the binary file of the firmware to be flashed and a text file describing the changed parameters.
To flash the firmware on your electric scooter, simply use one of the third-party software available on the AppStore or PlayStore and start testing. In our case, an iOS app still under development called Scooter Companion was used. In figure 4.9, a screenshot of the app is shown.

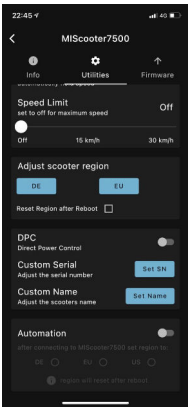


Figure 4.9. Scooter Companion Application

Chapter 5

Experimental results

Below we will illustrate numerous graphs aimed at showing the performance of the electric scooter with the standard firmware, driven without a wheelchair, and with the custom firmware created in this thesis, coupled to the wheelchair. Specifically, the graphs relating to current, battery discharge, temperature, residual distance, voltage and power supplied by the motor will be shown.

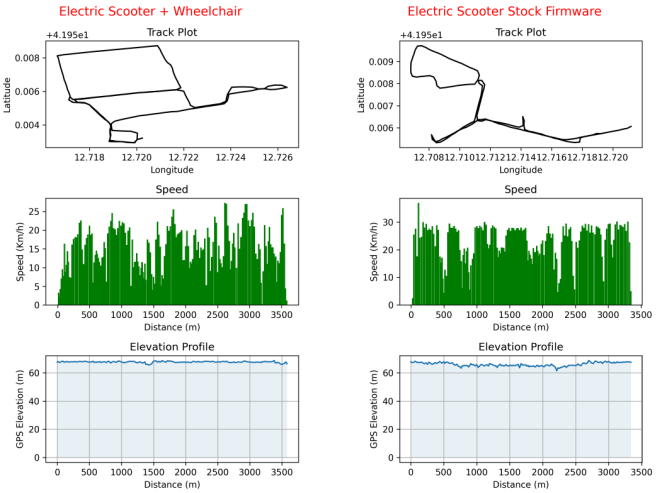


Figure 5.1. Distance traveled during the road test

As can be seen from the figure above, which shows 3 graphs relating to the position, speed and slope of the terrain, a road test lasting about 4 km was carried out on a mainly flat road at a speed as constant as possible.

From this point on we will show the graphs relating to the physical quantities that it was possible to acquire regarding the operation of the electric scooter.
In figure 5.2 it is possible to observe the trend in time of current and temperature and it can be seen that relatively to the first quantity there are positive peaks, related to the acceleration phases and negative peaks related to the braking phases in which the kinetic energy recovery system (KERS) takes over. It can be seen that the amplitude of these peaks is greater when the scooter is used with a wheelchair, since the moving mass is greater and therefore during the braking phase this system is able to recover a greater quantity of energy. With regard to the temperature, it can be seen that this is in a range between 28 and 34 degrees centigrade, a lower range than that observed in the canonical use of the scooter, and this is due to the way in which it was decided to map the law of acceleration, specifically, with the use of the wheelchair, the quadratic law was used compared to the classic linear law. This behavior therefore allows you to enjoy a pleasant driving experience, functional and effective, while preserving the electronic components, primarily the battery, a component capable of showing a highly variable behavior with respect to the temperature at which it operates. Therefore, operating at a lower temperature allows you to lengthen the life of the battery and also to extend its duration, understood as kilometers covered.

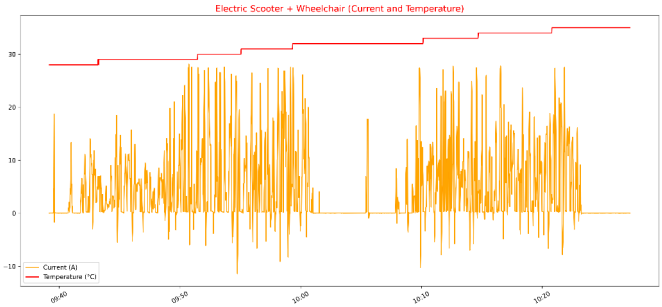


Figure 5.2. E-scooter + Wheelchair, current and temperature over time

Figure 5.3 shows the trend over time of the current and operating temperature in the standard use of the electric scooter equipped with stock firmware. it is possible to notice that there are many positive peaks as in the previous case but in this situation the negative peaks, therefore due to the recovery of kinetic energy, are of smaller amplitude due to the reduced mass. As far as the temperature is concerned, we can observe that this already in the initial phases is well above 35 degrees centigrade and how over time it reaches a temperature close to 40 degrees centigrade. As explained above, this significant difference is due to the different law that regulates acceleration and therefore the supply of current in the scooter. Specifically, the use of a linear law, the one present in the stock firmware, due to its nature provides for higher and extended current levels over time compared to the use of a quadratic or exponential law thus leading to higher operating temperatures.

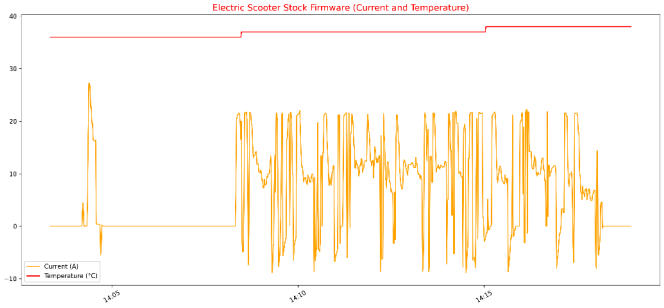


Figure 5.3. E-scooter, Current and temperature over time

The following graphs will show the trend over time of the battery discharge, of the remaining distance and of the available voltage. Specifically in figure 5.4 you can see these three trends in the case of using the electric scooter with custom firmware and coupled to a wheelchair and you can see how, due to the energy required by the motor to move a larger mass, the battery drain has a much steeper course than it would have with the stock firmware. In particular it can be noted that while in the canonical case the discharge presents a more regular trend over time, in the other case there are numerous charging and discharging phases or the steep discharges are compensated by the numerous times in which the KERS is able to recover the kinetic energy which in this case is greater. However, the energy requirement remains very high and the KERS can only slow down the relentless battery discharge which would have been much steeper without this energy recovery system.

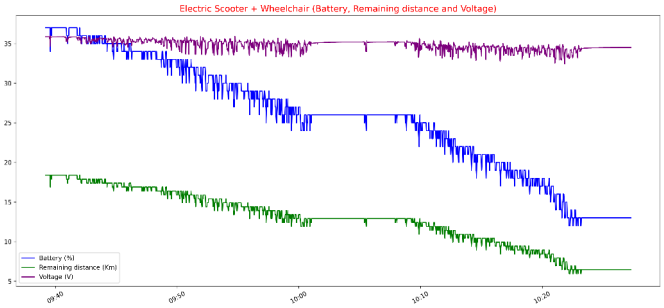


Figure 5.4. E-scooter + Wheelchair, Battery, remaining distance and voltage over time

In figure 5.5 we can observe the trends shown previously for use with the standard firmware and how this is able to guarantee a more contained use of the available energy, both for the reduced mass to be towed, and for the different law that adjusts the acceleration. In fact, we can notice that the battery discharge phase is softer and with peaks related to energy recovery of lesser amplitude and frequency than in the previous case since in this case a wheelchair is not towed.

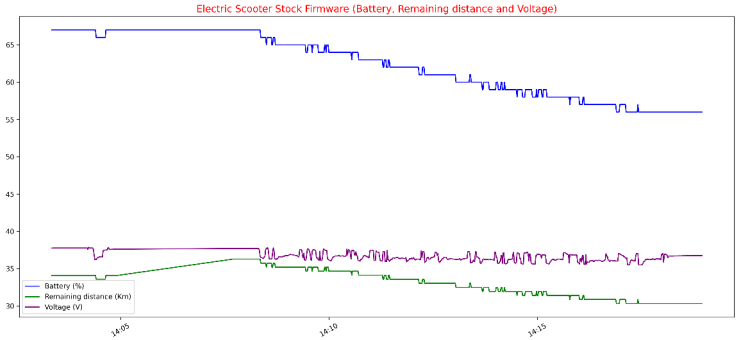


Figure 5.5. E-scooter, Battery, remaining distance and voltage over time

The following figures show the power delivered by the engine during the typical use experiment and in particular it can be noted that in the case of using the scooter with custom firmware and wheelchair the power peaks are much higher than in the case with stock firmware. This is necessary not only to allow the scooter to have a greater starting base than in the normal case but also to be able to carry out accelerations that allow the towing of a greater mass.

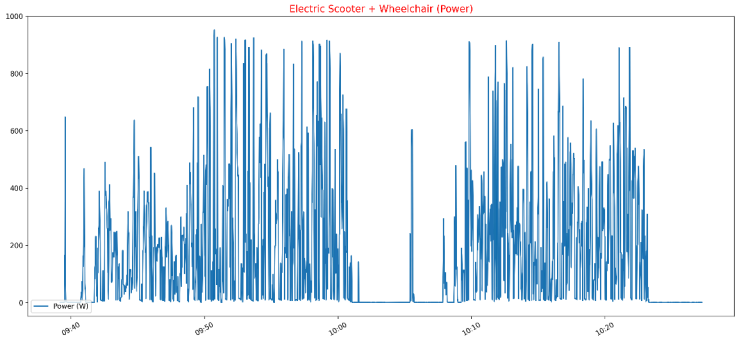


Figure 5.6. E-scooter + Wheelchair, Power over time

It can be seen that in case of stock firmware the average power level is 380W with peaks reaching 780W while using our custom firmware the average power output is about 550W with peaks in the order of 980W. These values are related only to one of the three driving modes of the electric scooter, sport mode since in the firmware customization phase it was decided to leave the other two modes unchanged to ensure longer battery life.

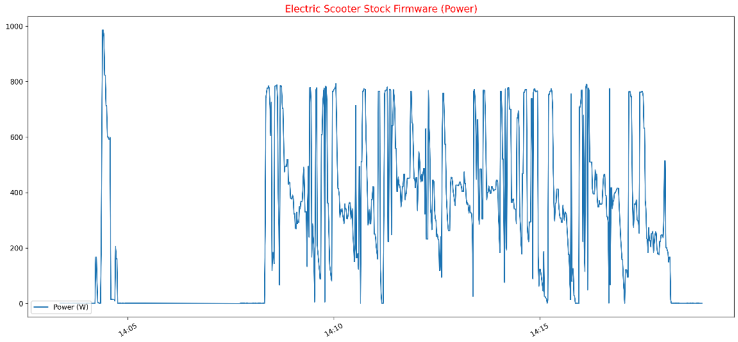


Figure 5.7. E-scooter, Power over time

Chapter 6

Conclusions and future works

The main objective of this thesis has been, net of small design issues, to demonstrate and realize the feasibility of having an economic solution that would allow to achieve all the objectives set, that is to demonstrate that the same means of locomotion can be used both by people with motor difficulties or disabilities and by people free from disabilities of various kinds. Moreover, this result was achieved without modifying in any way the structure of the electric scooter and the wheelchair, in fact it was necessary only the addition of a metal support to achieve the mechanical coupling between the two parts and the modification of the software side to allow the use of the scooter in this new area of use. It is important to note that once modified the firmware of the scooter to allow its use by people with motor difficulties, it can still be used without any problem even by people with normal mobility. In fact, the firmware despite having been modified by deactivating the KERS and modifying various parameters, does not affect the behavior and the quality of the movement expressed by the electric scooter, resulting perfectly compatible even in its standard use without causing problems of any kind.
Based on the experiments carried out, encouraging results have emerged. It was possible to obtain the mechanical coupling between the two vehicles and the modification of the firmware without affecting the operation of the electric scooter, thus eliminating consequent problems for the user. Moreover, this has been achieved without having to change the structure and nature of the electric scooter and the wheelchair, so as to ensure the general operation only through their physical connection.
On the other hand, less encouraging results also emerged from the experiments since, as shown in the chapter on mounting methods, both do not allow for 100% efficient and effective riding since both modes have advantages and limitations due to the physical structure of the two such heterogeneous means of transportation. In particular, the main defect related to reduced visibility would be addressed when there are electric scooters on the market with telescopic steering, which, however, are currently very uncommon and in any case a modification of the steering would require considerable adaptation work. This problem would therefore not allow the use of shared electric scooters, unless they are equipped with a retractable or folding handlebar. On the other hand, the second mode would allow the use of the electric scooter by both able-bodied and disabled users, but at the expense of a

reduced turning radius when using a wheelchair. Therefore, we believe that this second mode is the most suitable to be used on a permanent basis and would allow access to the benefits and potential of the electric scooter to the widest possible audience.
However, there is still much work to be done as this project could be extended to more brands of electric scooters and wheelchairs. Companies operating in the electric scooter rental market could also be involved to share with them the results obtained in this thesis work thus allowing disabled people to be able to use the modified firmware according to their needs, with a simple request to the server via an application.

6.1 Future Works

Further development could be aimed at obtaining a finer tuning of the firmware parameters and thus more effective and comfortable for driving by people with disabilities, while also adapting it to the various engine models available on the market. Further room for improvement can be achieved by integrating state of the art technologies into this project, for example through the use of machine learning for computer vision tasks. Specifically, video cameras could be integrated into the electric scooter and then used to be able to have a representation of the outside world and based on this ensure greater safety through the use of machine learning algorithms, for example in helping the user to perform the braking phase when an obstacle or red light is detected.
To conclude, further developments already planned include the possibility to install a seatbelt on the wheelchair, an increased battery that allows to extend the autonomy of the electric scooter and a reinforcement of its control unit that allows for more performance and faster motor reaction times. Reinforcing the control unit consists of adding tin and possibly copper wire to the back of the control unit itself, where high levels of current pass through. This will increase the diameter of the wire, which will prevent overheating. More extensive and advanced work may include replacing the mosfets [1] with higher quality components without causing any damage to the control unit with firmware using higher currents.
In addition, further future development will focus on performing engineering work on the fabricated prototype. For example, since the metal structure might be too expensive and at the same time too heavy, it might be possible to realize it in more modern materials, i.e. composite materials, or with plastic or aluminum derived materials that can be realized by 3D printing the AutoCAD project of the prototype shown in Appendix D. In addition, it could be possible through agreements with major companies to extend this product to a wide audience of people with mobility impairments thus allowing the renting of scooters by these people.

[1] The MOSFET, often known as MOS transistor, indicates a type of field-effect transistor widely used in the field of digital and analogue electronics.

Listings

List of Figures

Appendix A

Code used to create the custom firmware of the electric scooter

A.1 Project folder structure

The entire project of this thesis is contained within my *GitHub repository*. The code within it is divided into folders, subfolders and files according to the scheme described by the figure A.1.

- The *root* folder of the project contains the core of the project itself, i.e. modules that allow modification of individual parameters of the electric scooter firmware described in chapter 4. In addition to the subfolders in this directory we also find here test files used during the programming phase.
- The *draw* folder contains 4 python files to build graphs obtained from road tests with the scooter in the two driving modes previously described to highlight the differences between using original firmware or custom firmware to tow the wheelchair and allow driving a disabled person. In addition to the static graphs, dynamic graphs were also created to describe the route taken during the test phases, showing the road travelled, the speed and the altitude.
- The folder *enc_dec* contains files that are used to decrypt the firmware obtained by listening to http packets from the official application of the electric scooter and then encrypt it to flash the new custom firmware.
- The files in the folder *gui*, as you can understand from the name, allow the creation of the graphical interface that through the use of a single button automates the creation of custom firmware and its saving for flashing.

In order to use the code in the following repository you need to install the python3 libraries used by the project using the following terminal command:

```
~ sudo apt install python3
~ sudo apt install python3-pip
~ git clone https://github.com/LucaTomei/Electric-Scooter-Wheelchair
~ cd Electric-Scooter-Wheelchair
~ pip install -r "requirements.txt"
```

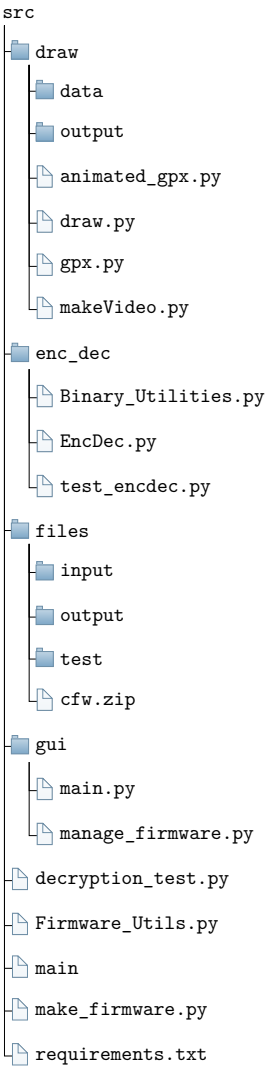


Figure A.1. GitHub repository structure

A.2 E-scooter firmware Encrytpion/Decryption classes

```
from struct import pack, unpack

class Binary_Utilities(object):
  def __init__(self):
    pass

  def xor(self, s1, s2):
    res = bytearray()
    for i in range(8):  res.append(s1[i] ^ s2[i])
    return res

  # check autenticity of the data
  def checksum(self, data):
    s = 0
    for i in range(0, len(data), 4):  s += unpack('<L', data[i:i+4])[0]
    return (((s >> 16) & 0xFFFF) | ((s & 0xFFFF) << 16)) ^ 0xFFFFFFFF


  # ECB
  # The simplest of the encryption modes is the electronic codebook (ECB) mode
  # (named after conventional physical codebooks[19]). The message is divided
    into blocks, and each block is encrypted separately.

  # ecb encryption
  def ecb_enc(self, block, key):
    y, z = unpack('<LL', block)
    k = unpack('<LLLL', key)
    s = 0

    for i in range(32):
      s = (s + 0x9E3779B9) & 0xFFFFFFFF
      y = (y + (((z << 4) + k[0]) ^ (z + s) ^ ((z >> 5) + k[1]))) & 0xFFFFFFFF
      z = (z + (((y << 4) + k[2]) ^ (y + s) ^ ((y >> 5) + k[3]))) & 0xFFFFFFFF
    return pack('<LL', y, z)

  # ecb decryption
  def ecb_dec(self, block, key):
    y, z = unpack('<LL', block)
    k = unpack('<LLLL', key)
    s = 0xC6EF3720

    for i in range(32):
      z = (z - (((y << 4) + k[2]) ^ (y + s) ^ ((y >> 5) + k[3]))) & 0xFFFFFFFF
      y = (y - (((z << 4) + k[0]) ^ (z + s) ^ ((z >> 5) + k[1]))) & 0xFFFFFFFF
      s = (s - 0x9E3779B9) & 0xFFFFFFFF
    return pack('<LL', y, z)



  # The data which will be encrypted must be 8 byte aligned!
    # We also have to write a checksum to the last 4 bytes.
    # Zero pad for 4-byte aligning first:
  def pad(self, data):
    sz = len(data)
    if sz % 4:
```

```
55      o = (4 - (sz % 4))
56      data += b'\x00' * o
57      sz += o
58
59    # If we're 8-byte aligned now then add 4 zero pad bytes
60    if (sz % 8) == 0: data += b'\x00\x00\x00\x00'
61
62    # so we can add our 4 checksum bytes and be 8-byte aligned
63    return data + pack('<L', self.checksum(data))
64
65  def unpad(self, data):
66    chk = unpack('<L', data[-4:])[0]
67    s = self.checksum(data[:-4])
68    assert s == chk, 'checksum does not match!'
69    return data[:-4]
```

Listing A.1. Binary_Utilities.py

```
# With this module you can easily encode or decode Xiaomi-Ninebot firmware files or make ZIP archives. It takes as input the binary file sniffed by the Ninebot / Xiaomi application and returns the decrypted file for the subsequent modification of the parameters.
# We can use the same module to encrypt the previously modified and decrypted file

from struct import pack, unpack

try:    from .Binary_Utilities import Binary_Utilities
except Exception as e:  from Binary_Utilities import Binary_Utilities

class EncDec(object):
  def __init__(self):
    self.Binary_Utilities_OBJ = Binary_Utilities()
    UPDKEY=b'\xFE\x80\x1C\xB2\xD1\xEF\x41\xA6\xA4\x17\x31\xF5\xA0\x68\x24\xF0'
    self.key = UPDKEY
    self.iv = b'\x00' * 8
    self.offset = 0

  def update_key(self):
    k = bytearray()
    for i in range(16): k.append((self.key[i] + i) & 0xFF)
    self.key = k

  def encrypt(self, data):
    data = self.Binary_Utilities_OBJ.pad(data)
    assert len(data) % 8 == 0, 'data must be 8 byte aligned!'
    res = bytearray()
    for i in range(0, len(data), 8):
      ct = self.Binary_Utilities_OBJ.ecb_enc(self.Binary_Utilities_OBJ
          .xor(self.iv, data[i:i+8]), self.key)
      res += ct
      self.iv = ct
      self.offset += 8
      if (self.offset % 1024) == 0: self.update_key()
    return res

  def decrypt(self, data):
    assert len(data) % 8 == 0, 'data must be 8 byte aligned!'
    res = bytearray()
    for i in range(0, len(data), 8):
      ct = data[i:i+8]
      res += self.Binary_Utilities_OBJ
          .xor(self.iv, self.Binary_Utilities_OBJ.ecb_dec(ct, self.key))
      self.iv = ct
      self.offset += 8
      if (self.offset % 1024) == 0: self.update_key()
    return self.Binary_Utilities_OBJ.unpad(res)

if __name__ == '__main__':
  EncDec_OBJ = EncDec()
```

Listing A.2. EncDec.py

```python
#!/usr/bin/python
from sys import argv, exit
from os.path import exists, getsize
from EncDec import EncDec

class Encryption_Test(object):
  def __init__(self):
    self.encdec_obj = EncDec()

  def make_test(self, choice):
    assert choice == 1 or choice == 2, "\n\n\tchoice can be 1 or 2"
    if choice == 1:
      print("Insert binary to Encrypt")
      input_filename = input()
      if not exists(input_filename):
        print("Input filename doesn't exists!")
        exit(1)

      print("Insert output filename")
      output_filename = input()

      hfi = open(input_filename, 'rb')
      hfo = open(output_filename + ".bin", 'wb')
      hfo.write(self.encdec_obj.encrypt(hfi.read()))
      hfo.close()
      hfi.close()
    else:
      print("Insert binary to Decrypt")
      input_filename = input()
      if not exists(input_filename):
        print("Input filename doesn't exists!")
        exit(1)

      print("Insert output filename (withou '.enc')")
      output_filename = input()

      hfi = open(input_filename, 'rb')
      hfo = open(output_filename + ".bin.enc", 'wb')
      hfo.write(self.encdec_obj.decrypt(hfi.read()))
      hfo.close()
      hfi.close()

  def main(self):
    print("[*] Encryption/Decryption Test[*]")
    print("\t[1] for Encryption\n\t[2] for Decryption")
    choice = input()
    if choice.isnumeric():
      self.make_test(int(choice))
    else:
      print("The choice made is not correct")

if __name__ == '__main__':
  Encryption_Test_OBJ = Encryption_Test()
  Encryption_Test_OBJ.main()
```

Listing A.3. test_encdec.py

A.3 E-scooter firmware parameters editor classes

```
# Signature exception
class SgnException(Exception):  pass

class Firmware_Utils(object):
  def __init__(self): pass

  def Patch(self, data, ofs, size, imm, signature):
    assert size % 2 == 0, 'size must be power of 2!'
    assert len(signature) == size * 8, 'signature must be size * 8 long!'
    imm = int.from_bytes(imm, 'little')
    sfmt = '<' + 'H' * (size // 2)

    sigs = [signature[i:i + 16][::-1] for i in range(0, len(signature), 16)]
    orig = data[ofs:ofs+size]
    words = struct.unpack(sfmt, orig)s
    patched = []
    for i, word in enumerate(words):
      for j in range(16):
        imm_bitofs = sigs[i][j]
        if imm_bitofs is None:  continue
        imm_mask = 1 << imm_bitofs
        word_mask = 1 << j
        if imm & imm_mask:  word |= word_mask
        else: word &= ~word_mask
      patched.append(word)

    packed = struct.pack(sfmt, *patched)
    data[ofs:ofs+size] = packed
    return (orig, packed)

    # To find the pattern
  def GetPattern(self, data, signature, mask=None, start=None, maxit=None):
    sig_len = len(signature)

    if start is None: start = 0
    stop = len(data)

    if maxit is not None: stop = start + maxit

    if mask:
      assert sig_len == len(mask), 'mask must be as long as the signature!'
      for i in range(sig_len):
        if signature[i] is not None:  signature[i] &= mask[i]

    for i in range(start, stop):
      matches = 0
      while signature[matches] is None or signature[matches] == (data[i +
    matches] & (mask[matches] if mask else 0xFF)):
        matches += 1
        if matches == sig_len:  return i

    raise SgnException('Pattern not found!')
```

Listing A.4. Firmware_Utils.py

```python
#!/usr/bin/python3
from binascii import hexlify
import struct, keystone, sys
from enc_dec import EncDec

from Firmware_Utils import Firmware_Utils

# https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/readings/ARMv7-M_ARM.pdf
MOVW_T3_IMM = [*[None]*5, 11, *[None]*6, 15, 14, 13, 12, None, 10, 9, 8, *[None]*4, 7, 6, 5, 4, 3, 2, 1, 0]
MOVS_T1_IMM = [*[None]*8, 7, 6, 5, 4, 3, 2, 1, 0]


## Firmware Patcher Main class
class Make_Firmware():
    def __init__(self, data):
      self.Firmware_Utils_OBJ = Firmware_Utils()
      self.data = bytearray(data)

        # Keystone is a lightweight multi-platform, multi-architecture assembler framework.  Keystone compile in this case arch arm assembly instruction to thumb (thumbs are set of 16 bit instructions)
        # https://gist.github.com/aquynh/d7cf8788b4e16a8c3078
      self.ks = keystone.Ks(keystone.KS_ARCH_ARM, keystone.KS_MODE_THUMB)

    def encrypt(self):  self.data = EncDec.EncDec().encrypt(self.data)
    def decrypt(self):  self.data = EncDec.EncDec().decrypt(self.data)

    # This algorithm replace the linear throttle curve with a quadratic one.
    # With the quadratic curve, the power delivery works on a logarithmic scale, sweet at the low end and then soaring on the final.
    def throttle_alg(self):
        sig = [0xF0, 0xB5, 0x25, 0x4A, 0x00, 0x24, 0xA2, 0xF8, 0xEC, 0x40, 0x24, 0x49]
        ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 4
        pre, post = self.data[ofs:ofs + 1], bytearray((0x01, 0x24))
        self.data[ofs:ofs + 2] = post
        return [(ofs, pre, post)]

    # Disable electric scooter automatic DRV updates on firmware changes
    def version_spoofing(self):
        sig = [0x4F, 0xF4, 0x93, 0x70, 0xA0, 0x86, 0x12, 0x48, 0x00, 0x78]
        ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig)
        pre = self.data[ofs:ofs+4]
        post = bytes(self.ks.asm('MOVW   R0, #0x526')[0])
        self.data[ofs:ofs+4] = post
        return [(ofs, pre, post)]

  # Minimum speed below which the kers is activated
    def kers_min_speed(self, kmh):
        val = struct.pack('<H', int(kmh * 390))
        sig = [0x25, 0x68, 0x40, 0xF6, 0x24, *[None]*2, 0x42]
        ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 2
        pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 4, val, MOVW_T3_IMM)
        return [(ofs, pre, post)]
```

```
51
52      # Motor start speed
53      def motor_start_speed(self, kmh):
54          ret = []
55          val = int(kmh * 390)
56          val = val - (val % 16)
57          assert val.bit_length() <= 12, 'bit length overflow'
58          sig = [0x4B, 0x01, None, None, 0x48, 0x00, None, None, None, 0xB6, 0xF5,
     0xC5, 0x6F, 0x0D, 0xDB]
59          ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 9
60          pre = self.data[ofs:ofs + 4]
61          post = bytes(self.ks.asm('CMP.W R6, #{:n}'.format(val))[0])
62          self.data[ofs:ofs + 4] = post
63          ret.append((ofs, pre, post))
64          ofs += 4
65
66          pre, post = self.data[ofs:ofs + 1], bytearray((0x0D, 0xDB))
67          self.data[ofs:ofs + 2] = post
68          ret.append((ofs, pre, post))
69          return ret
70
71      # If you plan to use the Ninebot Max G30 lock functionality from the
        application and you do not want the scooter to turn off after a few hours,
        you need to flag this field.
72      def stay_on_locked(self):
73          sig = [0x03, 0xF0, None, None, 0xF0, 0x7B]
74          ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig)
75          pre = self.data[ofs:ofs+4]
76          post = bytes(self.ks.asm('NOP;NOP')[0])
77          self.data[ofs:ofs+4] = post
78          return [(ofs, pre, post)]
79
80      # This option removes the ignition when connecting the charger and allows
        you to connect additional batteries in parallel to increase autonomy.
81      def remove_charging_mode(self):
82          sig = [0x20, 0xB1, 0x84, 0xF8, 0x38, 0x60, 0xE0, 0x7B, 0x18, 0xB1]
83          ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig)
84          pre = self.data[ofs:ofs+2]
85          post = bytes(self.ks.asm('NOP')[0])
86          self.data[ofs:ofs+2] = post
87          return [(ofs, pre, post)]
88
89
90      # Speed parameters
91      def speed_params(self, normal_kmh, normal_phase, normal_battery):
92          ret = []
93          sig = [0x95, 0xF8, 0x4F, 0x80, 0x4D, 0xF2, 0xD8, 0x63, 0xB8, 0xF1, 0x01,
     0x0F, 0x0A, 0xD0]
94          ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 4
95          pre = self.data[ofs:ofs+4]
96          post = bytes(self.ks.asm('MOVW R3, #{:n}'.format(normal_phase))[0])
97          self.data[ofs:ofs+4] = post
98          ret.append([ofs, pre, post])
99
100         sig = [0x46, 0xF2, 0xA8, 0x12, 0xFB, 0xB1, 0x22, 0xE0, 0x95, 0xF8]
101         ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig)
102         pre = self.data[ofs:ofs+4]
```

```
103         post = bytes(self.ks.asm('MOVW R2, #{:n}'.format(normal_battery))[0])
104         self.data[ofs:ofs+4] = post
105         ret.append([ofs, pre, post])
106 
107         sig = [0x90, 0x42, 0x01, 0xD2, 0xE0, 0x85, 0x00, 0xE0, 0xE2, 0x85, 0x21
    ]
108         ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig)
109         pre = self.data[ofs:ofs+2]
110         post = bytes(self.ks.asm('CMP R2, R2')[0])
111         self.data[ofs:ofs+2] = post
112         ret.append([ofs, pre, post])
113 
114         ofs += 10
115         pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 2, struct.
    pack('<B', normal_kmh), MOVS_T1_IMM)
116         ret.append([ofs, pre, post])
117         self.data[ofs:ofs+2] = post
118         ret.append([ofs, pre, post])
119 
120         return ret
121 
122 
123     # Motor Power Constant represents the ability of the motor to convert
        electrical power
124     # into mechanical power. The motor constant is given by:
125     #   k_m = T / \sqrt(P), where
126     #  - k_m = motor constant (Nm / \sqrt(Watt))
127     #  - T = torque (Nm)
128     #  - P = resistive power losses (also known as I^2\cdot R losses) (W)
129     # therefore The Motor Power constant is a value that increases the power
        delivered in an inversely proportional way to the set number, ie the lower
        this number the higher the power will be.
130 
131     # lower value = more power
132     # original = 51575 (~500 Watt)
133     # DYoC = 40165 (~650 Watt)
134     # CFW W = 27877 (~850 Watt)
135     # CFW = 25787 (~1000 Watt)
136     def mpc(self, val):
137         val = struct.pack('<H', int(val))
138         ret = []
139         sig = [0x31, 0x68, 0x2A, 0x68, 0x09, 0xB2, 0x09, 0x1B, 0x12, 0xB2, 0xD3,
     0x1A, 0x4C, 0xF6, 0x77, 0x12]
140         ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 12
141         pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 4, val,
    MOVW_T3_IMM)
142         ret.append((ofs, pre, post))
143         ofs += 4
144 
145         ofs += 4
146         pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 4, val,
    MOVW_T3_IMM)
147         ret.append((ofs, pre, post))
148 
149         sig = [0xD3, 0x1A, 0x4C, 0xF6, 0x77, 0x12]
150         ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig, None, ofs,
    100) + 2
```

```
151         pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 4, val,
    MOVW_T3_IMM)
152         ret.append((ofs, pre, post))
153         ofs += 4
154 
155         ofs += 4
156         pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 4, val,
    MOVW_T3_IMM)
157         ret.append((ofs, pre, post))
158 
159         sig = [0xC9, 0x1B, 0x4C, 0xF6, 0x77, 0x13]
160         ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig, None, ofs,
    100) + 2
161         pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 4, val,
    MOVW_T3_IMM)
162         ret.append((ofs, pre, post))
163         return ret
164 
165     def bypass_BMS(self):
166         sig = [0x06, 0x49, 0x4A, 0x78, 0x82, 0x42, 0x02, 0xD8, 0x4A, 0x78]
167         ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig)
168         pre = self.data[ofs:ofs+2]
169         post = bytes(self.ks.asm('BX LR')[0])
170         self.data[ofs:ofs+2] = post
171         return [(ofs, pre, post)]
172 
173 
174     # After how many seconds at constant speed the cruise control will start
    working, the standard value is 5 seconds and can be set in a range from 1
    to 10 seconds.
175     def cc_delay(self, delay):
176         delay = int(delay * 200)
177         assert delay.bit_length() <= 12, 'bit length overflow'
178         sig = [0x48, 0xB0, 0xF8, 0xF8, None, 0x33, 0x4A, 0x4F, 0xF4, 0x7A, 0x71,
     0x01, 0x28]
179         mask = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xF8, 0xFE, 0xFF, 0xFF, 0xFF, 0
    xFE, 0xFF, 0xFE]
180         ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig, mask) + 7
181         pre = self.data[ofs:ofs+4]
182         post = bytes(self.ks.asm('MOV.W R1, #{:n}'.format(delay))[0])
183         self.data[ofs:ofs+4] = post
184         return [(ofs, pre, post)]
185 
186 
187     # Maximum speed value
188     def max_speed(self, kmh):
189         ret = []
190         val = struct.pack('<B', int(kmh))
191         sig = [None, 0xF8, 0x34, 0xC0, None, None, 0x43, 0xF2]
192         ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 4
193         pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 2, val,
    MOVS_T1_IMM)
194         ret.append((ofs, pre, post))
195 
196         sig = [None, None, None, None, 0x17, None, None, 0x83, 0x46, 0xF6, 0x60
    ]
197         ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 4
```

```
198         pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 2, val,
        MOVS_T1_IMM)
199
200         ret.append((ofs, pre, post))
201
202         if self.data[0x7BAA] == 0x33 and self.data[0x7BAB] == 0x11:
203             sig = [None, 0xF8, 0x2E, 0xE0, 0x22, 0x20, 0xE0, 0x83, 0x1B, 0xE0,
        0x95]
204             ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 4
205             pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 2, val,
        MOVS_T1_IMM)
206         elif (self.data[0x8242] == 0xA8 and self.data[0x8243] == 0x71) or \
207              (self.data[0x8246] == 0x51 and self.data[0x8247] == 0x11):
208             sig = [0xDE, 0xD0, 0x11, 0xE0, 0x22, None, None, 0x83, 0xDE, 0xE7,
        0x95]
209             ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 4
210             pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 2, val,
        MOVS_T1_IMM)
211         else:
212             sig = [0x52, 0xC0, 0x4F, 0xF0, 0x22, 0x0E, 0x4C, 0xF2, 0x50, None,
        None]
213             ofs = self.Firmware_Utils_OBJ.GetPattern(self.data, sig) + 4
214             pre, post = self.Firmware_Utils_OBJ.Patch(self.data, ofs, 2, val,
        MOVS_T1_IMM)
215
216         ret.append((ofs, pre, post))
217         return ret
218
219
220 if __name__ == "__main__":
221   modality = 0
222   if len(sys.argv) != 3:
223     default_input_file = "files/input/DRV126.bin"
224     default_output_file = "files/output/FIRM.bin.enc"
225     to_print = "[*] Automatic mode [*]\n- Input file = %s\n - Output file = %s\
        n\n\t Custom Firmware Created Successfully!" %(default_input_file,
        default_output_file)
226     print(to_print)
227
228     with open(default_input_file, 'rb') as fp:  data = fp.read()
229   else:
230     modality = 1
231     with open(sys.argv[1], 'rb') as fp: data = fp.read()
232
233   cfw = Make_Firmware(data)
234   cfw.motor_start_speed(6)
235   cfw.cc_delay(2)
236   cfw.throttle_alg()
237   cfw.stay_on_locked()
238   cfw.encrypt()
239
240   output_filename = default_output_file if modality == 0 else sys.argv[2]
241   with open(output_filename, 'wb') as fp: fp.write(cfw.data)
```

Listing A.5. make_firmware.py

Appendix B

Code for the GUI of the custom firmware automatic generator

In order to automate the creation of custom firmware for the electric scooter with the relative modification of parameters to facilitate the driving of a disabled person, it was decided to create a simple and intuitive graphical interface that, by pressing two buttons, saves directly on the end user's computer the firmware that will then be flashed on the electric scooter.
The classes used for this purpose are illustrated in this appendix.

```
import requests, os, zipfile
from make_firmware import Make_Firmware

class Manage_Firmware(object):
  def __init__(self):
    self.download_dir = os.path.join(os.path.expanduser('~'), 'downloads')
    self.zip_filename = self.download_dir + "/" + "cfw.zip"

    self.cfw_dir = self.download_dir + "/cfw"

    self.params = """
      - Firmware Version: DRV126
      - Max speed: 25
      - Nominal draw: 33.0A
      - Max current: 55.0A
      - Stay on when scooter is locked
      - Wheel size: 10.0 inches
      - NO KERS
      - Throttle current coefficient: 1800
      - NO Cruise Control
      - Motor start speed: 0 km/h
      - Version spoofing
      """

    self.ninebot_jpeg = "ninebot.jpeg"

  def write_bin_file(self, filename, content):
    file = open(filename, "wb")
    file.write(content)
```

```
30    file.close()
31
32
33  def extract_zip_file(self, filename, extract_to):
34    with zipfile.ZipFile(filename, 'r') as zip_ref: zip_ref.extractall(
      extract_to)
35
36  def make_folder(self, folder_name):
37    if os.path.exists(folder_name): os.system("rm -rf " + folder_name)
38    os.mkdir(folder_name)
39
40  def remove_tmp_files(self):
41    os.system("rm -rf " + self.cfw_dir + "/*.txt")
42    os.system("rm -rf " + self.cfw_dir + "/*.json")
43
44
45  def write_txt_file(self, filename, content):
46    file = open(filename, "w")
47    file.write(content)
48    file.close()
49
50  def make_params_file(self, params_dir):
51    self.write_txt_file(params_dir + "/params.txt", self.params)
52
53
54  def make_zip_file(self, path):
55    ziph = zipfile.ZipFile(self.zip_filename, 'w', zipfile.ZIP_DEFLATED)
56    for root, dirs, files in os.walk(path):
57      for file in files:
58        ziph.write(os.path.join(root, file),
59          os.path.relpath(os.path.join(root, file),
60            os.path.join(path, '..')))
61    ziph.close()
62
63
64  def main(self):
65    if not os.path.exists("../files/cfw.zip"):
66      if os.path.exists(self.zip_filename): os.remove(self.zip_filename)
67      self.make_folder(self.cfw_dir)
68      self.Make_firmware.make_firmware()
69      self.extract_zip_file(self.zip_filename, self.cfw_dir)
70      self.remove_tmp_files()
71      self.make_params_file(self.cfw_dir)
72
73      os.remove(self.zip_filename)
74
75      self.make_zip_file(self.cfw_dir)
76    else:
77      os.system("cp ../files/cfw.zip " + self.download_dir)
78
79    if os.path.exists(self.cfw_dir):  os.system("rm -rf " + self.cfw_dir)
80    if os.path.exists(self.ninebot_jpeg): os.remove(self.ninebot_jpeg)
```

Listing B.1. manage_firmware.py

```
import tkinter as tk, PIL.Image, PIL.ImageTk, requests, time
from tkinter import font
from manage_firmware import Manage_Firmware

cfw_obj = Manage_Firmware()

HEIGHT = 550
WIDTH = 660

def download_bg_image():
    url = "https://www.impactsurf.com/images/thumbs/002/0025731_ninebot-kickscooter-max-g30-powered_870.jpeg"
    response = requests.get(url)
    content = response.content
    cfw_obj.write_bin_file(cfw_obj.ninebot_jpeg, content)



update_status = 0

root = tk.Tk()

def center_window(win):
    win.update_idletasks()
    width = win.winfo_width()
    frm_width = win.winfo_rootx() - win.winfo_x()
    win_width = width + 2 * frm_width
    height = win.winfo_height()
    titlebar_height = win.winfo_rooty() - win.winfo_y()
    win_height = height + titlebar_height + frm_width
    x = win.winfo_screenwidth() // 2 - win_width // 2
    y = win.winfo_screenheight() // 2 - win_height // 2
    win.geometry('{}x{}+{}+{}'.format(width, height, x, y))
    win.deiconify()


def destroy():
    root.destroy()


def download_custom_firmware():
    api_link = ""


def get_firmware_informations():
    global update_status
    if update_status == 0:
        to_print = cfw_obj.params
        cfw_obj.main()
        to_print = to_print + "\n\n\n" + "Generated firmware is in Download directory.\nClick the button above to exit"
        label.config(font=('Courier', 14))
        label['text'] = to_print
        update_status = 1
        button["highlightbackground"] = "red"
        button["fg"] = "white"
        button["text"] = "EXIT"
```

```
    else:
        button["state"] = "disabled"
        button["highlightbackground"] = "grey"
        to_print = "Bye Bye"
        label['text'] = to_print
        label.after(2000, destroy)


download_bg_image()
canvas = tk.Canvas(root, height=HEIGHT, width=WIDTH)
canvas.pack()

im = PIL.Image.open(cfw_obj.ninebot_jpeg)
photo = PIL.ImageTk.PhotoImage(im)
background_label = tk.Label(root, image=photo)
background_label.place(relx=0, rely=0, relwidth=1, relheight=1)

frame = tk.Frame(root, bg="#87cefa", bd=5)
frame.place(relx=0.5, rely=0.1, relwidth=0.75, relheight=0.1, anchor='n')


button = tk.Button(frame, text="Get Modified Custom Firmware", bg="gray", fg="
    white", font=('Courier', 14), command=lambda: get_firmware_informations())
button.place(relx=0, rely=0, relwidth=1, relheight=1)


lower_frame = tk.Frame(root, bg='#90c1ff', bd=10)
lower_frame.place(relx=0.5, rely=0.25, relwidth=0.75, relheight=0.6, anchor='n'
    )

label = tk.Label(lower_frame, font=('Courier', 18))
label.place(relx=0, rely=0, relwidth=1, relheight=1)

label["text"] = "Click the buttton above to\ngenerate modified custom firmware.
    "


# Center window
root.eval('tk::PlaceWindow %s center' % root.winfo_pathname(root.winfo_id()))
root.title('Custom Firmware Generator')

root.mainloop()
```

Listing B.2. gui_main.py

Appendix C

Code used to generate the graphs relating to the various road tests

During the road tests carried out both with the electric scooter assembled to the wheelchair with the custom firmware and with the only electric scooter equipped with original firmware it was decided to collect data related to the route taken, speed, current, battery, voltage, power, acceleration and remaining distance. Once these data were obtained in .csv and .gpz format using the iOS application DarknessBot, they were processed to obtain a graphical representation using Python classes collected in this appendix.

```
import datetime, random, matplotlib.pyplot as plt, matplotlib.dates as mdates

def getFileContent(filename):
  file = open(filename, "r")
  content = file.read()
  file.close()
  return content

def parseDate(datestr):
  "09.08.2021 10:28:17.542"
  datestr = ".".join(datestr.split(".")[:3])
  return datetime.datetime.strptime(datestr, "%d.%m.%Y %H:%M:%S")

def getCSVTuple(content):
  list1, list2 = []
  content = content.split("\n")
  for i in content:
    if ',' in i:
      item = i.split(",")
      if item[0] != "" or item[1] != "":
        if "." in item[0]:
          date = parseDate(item[0])
          list1.append(date)
        else:
          pass
          #list1.append(item[0])
        if "." in item[1]:
          list2.append(float(item[1]))
  return list1, list2
```

```
30
31 def mergeDates(batt1, corr1, dr1, pot1, temp1, volt1):
32   content_batt1 = getFileContent(batt1)
33   b1l1, b1l2 = getCSVTuple(content_batt1)
34
35   content_corr1 = getFileContent(corr1)
36   c1l1, c1l2 = getCSVTuple(content_corr1)
37
38   content_dr1 = getFileContent(dr1)
39   d1l1, d1l2 = getCSVTuple(content_dr1)
40
41   content_pot1 = getFileContent(pot1)
42   p1l1, p1l2 = getCSVTuple(content_pot1)
43
44   content_temp1 = getFileContent(temp1)
45   t1l1, t1l2 = getCSVTuple(content_temp1)
46
47   content_volt1 = getFileContent(volt1)
48   v1l1, v1l2 = getCSVTuple(content_volt1)
49
50   iteration_len = min(len(b1l2), len(c1l1))
51   iteration_len = min(iteration_len, len(d1l1))
52   iteration_len = min(iteration_len, len(p1l1))
53   iteration_len = min(iteration_len, len(t1l1))
54   iteration_len = min(iteration_len, len(v1l1))
55
56   b1l1o, b1l2o, c1l1o, c1l2o, d1l1o, d1l2o, p1l1o, p1l2o, t1l1o, t1l2o, v1l1o,
       v1l2o = []
57
58   for i in range(iteration_len):
59     if b1l1[i] in c1l1 and b1l1[i] in d1l1 and b1l1[i] in p1l1 and b1l1[i] in
       t1l1 and b1l1[i] in v1l1 and b1l1[i] not in b1l1o:
60       b1l1o.append(b1l1[i])
61       b1l2o.append(b1l2[i])
62       c1l1o.append(c1l1[i])
63       c1l2o.append(c1l2[i])
64       d1l1o.append(d1l1[i])
65       d1l2o.append(d1l2[i])
66       p1l1o.append(p1l1[i])
67       p1l2o.append(p1l2[i])
68       t1l2o.append(t1l2[i])
69       t1l1o.append(t1l1[i])
70       v1l1o.append(v1l1[i])
71       v1l2o.append(v1l2[i])
72   return b1l1o, b1l2o, c1l1o, c1l2o, d1l1o, d1l2o, p1l1o, p1l2o, t1l1o, t1l2o,
       v1l1o, v1l2o
73
74
75 def makeXTicks(a_list):
76   ret = []
77   ret.append(a_list[0])
78   ret.append(a_list[-1])
79   length = len(a_list)
80
81   middle = int(len(a_list) / 2)
82   ret.append(a_list[middle])
83
```

```
  first_half = a_list[:len(a_list)//2]
  second_half = a_list[len(a_list)//2:]

  tmp = int(len(first_half) / 2)
  ret.append(a_list[tmp])
  # tmp = int(len(second_half) / 4)
  # ret.append(a_list[tmp])

  ret.append(datetime.datetime(2021, 8, 9, 10, 15, 37))
  return ret


def drawPlot(master_title, index, batt_file, corr_file, dr_file, pot_file,
    temp_file, volt_file):
  index = "output/"+index
  b1l1o, b1l2o, c1l1o, c1l2o, d1l1o, d1l2o, p1l1o, p1l2o, t1l1o, t1l2o, v1l1o,
    v1l2o = mergeDates(batt_file, corr_file, dr_file, pot_file, temp_file,
    volt_file)

  # PLOT 1
  plt.rcParams["legend.loc"] = 'lower left' # positioning legend
  plt.rcParams["figure.figsize"] = (15,7) # plot size

  ax = plt.gca()

  fig = plt.figure()
  plt.plot(c1l1o, c1l2o, label="Current (A)", color="orange")
  plt.plot(t1l1o, t1l2o, label="Temperature (C)", color="red")

  x_ticks = makeXTicks(b1l1o)
  plt.xticks(rotation=30)
  ax.set_xticks(x_ticks)

  plt.legend()
  dtFmt = mdates.DateFormatter("%H:%M") # define the formatting
  plt.gca().xaxis.set_major_formatter(dtFmt)

  #plt.show()
  plt.title(master_title + " (Current and Temperature)", fontsize=14, color="
    red")
  plt.tight_layout()
  plot_name = index +'_curr+temp.png'
  fig.savefig(plot_name, dpi=300)

  # PLOT 2
  plt.rcParams["legend.loc"] = 'lower left' # positioning legend
  plt.rcParams["figure.figsize"] = (15,7) # plot size

  ax = plt.gca()

  fig = plt.figure()
  plt.plot(b1l1o, b1l2o, label="Battery (%)", color="blue")
  plt.plot(d1l1o, d1l2o, label="Remaining distance (Km)", color="green")
  plt.plot(v1l1o, v1l2o, label="Voltage (V)", color="purple")

  x_ticks = makeXTicks(b1l1o)
  plt.xticks(rotation=30)
```

```
137   ax.set_xticks(x_ticks)
138
139   plt.legend()
140   dtFmt = mdates.DateFormatter("%H:%M") # define the formatting
141   plt.gca().xaxis.set_major_formatter(dtFmt)
142
143   #plt.show()
144   plt.title(master_title + " (Battery, Remaining distance and Voltage)",
        fontsize=14, color="red")
145   plt.tight_layout()
146   plot_name = index + '_batt+dist+volt.png'
147   fig.savefig(plot_name, dpi=300)
148
149   # PLOT 3
150   plt.rcParams["legend.loc"] = 'lower left' # positioning legend
151   plt.rcParams["figure.figsize"] = (15,7) # plot size
152
153
154   ax = plt.gca()
155
156   fig = plt.figure()
157
158   plt.plot(p1l1o, p1l2o, label="Power (W)")
159
160   x_ticks = makeXTicks(b1l1o)
161   plt.xticks(rotation=30)
162   ax.set_xticks(x_ticks)
163
164   plt.legend()
165   dtFmt = mdates.DateFormatter("%H:%M") # define the formatting
166   plt.gca().xaxis.set_major_formatter(dtFmt)
167
168   #plt.show()
169   plt.title(master_title + " (Power)", fontsize=14, color="red")
170   plt.tight_layout()
171
172   plot_name = index + '_power.png'
173   fig.savefig(plot_name, dpi=300)
174
175
176 def main():
177   # Electric Scooter & Wheelchair
178   batt1 = "/Users/lucasmac/Downloads/draw/data/2_carrozzina/Batteria (1 ora).
        csv"
179   corr1 = "/Users/lucasmac/Downloads/draw/data/2_carrozzina/Corrente (1 ora).
        csv"
180   dr1 = "/Users/lucasmac/Downloads/draw/data/2_carrozzina/Distanza Rimanente (1
         ora).csv"
181   pot1 = "/Users/lucasmac/Downloads/draw/data/2_carrozzina/Potenza (1 ora).csv"
182   temp1 = "/Users/lucasmac/Downloads/draw/data/2_carrozzina/Temperatura (1 ora).
        csv"
183   volt1 = "/Users/lucasmac/Downloads/draw/data/2_carrozzina/Voltaggio (1 ora).
        csv"
184   master_title = "Electric Scooter + Wheelchair"
185   index = "1"
186   drawPlot(master_title, index, batt1, corr1, dr1, pot1, temp1, volt1)
187
```

```
188
189   # Only Electric Scooter
190   batt1 = "/Users/lucasmac/Downloads/draw/data/only_scooter/Batteria (1 ora).
        csv"
191   corr1 = "/Users/lucasmac/Downloads/draw/data/only_scooter/Corrente (1 ora).
        csv"
192   dr1 = "/Users/lucasmac/Downloads/draw/data/only_scooter/Distanza Rimanente (1
         ora).csv"
193   pot1 = "/Users/lucasmac/Downloads/draw/data/only_scooter/Potenza (1 ora).csv"
194   temp1 = "/Users/lucasmac/Downloads/draw/data/only_scooter/Temperatura (1 ora).
        csv"
195   volt1 = "/Users/lucasmac/Downloads/draw/data/only_scooter/Voltaggio (1 ora).
        csv"
196   master_title = "Electric Scooter Stock Firmware"
197   index = "2"
198   drawPlot(master_title, index, batt1, corr1, dr1, pot1, temp1, volt1)
199
200   #print(b112)
201
202 if __name__ == '__main__':
203   main()
```

Listing C.1. draw_graphs.py

```
1  import matplotlib.pyplot as plt
2  import time
3  from IPython import display
4  from xml.dom import minidom
5  import math
6
7  def drawGPX(master_title, filename, outfilename):
8      #READ GPX FILE
9      data=open(filename)
10     xmldoc = minidom.parse(data)
11     track = xmldoc.getElementsByTagName('trkpt')
12     elevation=xmldoc.getElementsByTagName('ele')
13     datetime=xmldoc.getElementsByTagName('time')
14     n_track=len(track)
15
16     #PARSING GPX ELEMENT
17     lon_list=[]
18     lat_list=[]
19     h_list=[]
20     time_list=[]
21     for s in range(n_track):
22         lon,lat=track[s].attributes['lon'].value,track[s].attributes['lat'].
       value
23         elev=elevation[s].firstChild.nodeValue
24         lon_list.append(float(lon))
25         lat_list.append(float(lat))
26         h_list.append(float(elev))
27         # PARSING TIME ELEMENT
28         dt=datetime[s].firstChild.nodeValue
29         time_split=dt.split('T')
30         hms_split=time_split[1].split(':')
31         time_hour=int(hms_split[0])
32         time_minute=int(hms_split[1])
```

```
        time_second=int(hms_split[2].split('Z')[0])
        total_second=time_hour*3600+time_minute*60+time_second
        time_list.append(total_second)

    #GEODETIC TO CARTERSIAN FUNCTION
    def geo2cart(lon,lat,h):
        a=6378137 #WGS 84 Major axis
        b=6356752.3142 #WGS 84 Minor axis
        e2=1-(b**2/a**2)
        N=float(a/math.sqrt(1-e2*(math.sin(math.radians(abs(lat)))**2)))
        X=(N+h)*math.cos(math.radians(lat))*math.cos(math.radians(lon))
        Y=(N+h)*math.cos(math.radians(lat))*math.sin(math.radians(lon))
        return X,Y

    #DISTANCE FUNCTION
    def distance(x1,y1,x2,y2):
        d=math.sqrt((x1-x2)**2+(y1-y2)**2)
        return d

    #SPEED FUNCTION
    def speed(x0,y0,x1,y1,t0,t1):
        d=math.sqrt((x0-x1)**2+(y0-y1)**2) *3.6
        delta_t=t1-t0
        s=float(d/delta_t)
        return s


    #POPULATE DISTANCE AND SPEED LIST
    d_list=[0.0]
    speed_list=[0.0]
    l=0
    for k in range(n_track-1):
        if k<(n_track-1):
            l=k+1
        else:
            l=k
        XY0=geo2cart(lon_list[k],lat_list[k],h_list[k])
        XY1=geo2cart(lon_list[l],lat_list[l],h_list[l])

        #DISTANCE
        d=distance(XY0[0],XY0[1],XY1[0],XY1[1])
        sum_d=d+d_list[-1]
        d_list.append(sum_d)

        #SPEED
        s=speed(XY0[0],XY0[1],XY1[0],XY1[1],time_list[k],time_list[l])
        speed_list.append(s)


    #PLOT TRACK
    f,(track,speed,elevation)=plt.subplots(3,1)
    f.suptitle(master_title, fontsize=14, color="red")
    f.set_figheight(8)
    f.set_figwidth(5)
    plt.subplots_adjust(hspace=0.5)
    track.plot(lon_list,lat_list,'k')
    track.set_ylabel("Latitude")
```

```
    track.set_xlabel("Longitude")
    track.set_title("Track Plot")

    #PLOT SPEED
    speed.bar(d_list,speed_list,10,color='w',edgecolor='w')
    speed.set_title("Speed")
    speed.set_xlabel("Distance (m)")
    speed.set_ylabel("Speed (Km/h)")

    #PLOT ELEVATION PROFILE
    base_reg=0
    elevation.plot(d_list,h_list)
    elevation.fill_between(d_list,h_list,base_reg,alpha=0.1)
    elevation.set_title("Elevation Profile")
    elevation.set_xlabel("Distance (m)")
    elevation.set_ylabel("GPS Elevation (m)")
    elevation.grid()


    #ANIMATION/DYNAMIC PLOT
    for i in range(n_track):
        track.plot(lon_list[i],lat_list[i])
        speed.bar(d_list[i],speed_list[i],10,color='g',edgecolor='g')
        elevation.plot(d_list[i],h_list[i])
        display.display(plt.gcf())
        display.clear_output(wait=True)
        plt.pause(1)
        #time.sleep(1)
    #plt.show()
    plt.tight_layout()
    f.savefig(outfilename, dpi=300)

if __name__ == '__main__':
    master_title = "Electric Scooter + Wheelchair"
    filename = 'data/2_carrozzina/DarknessBot 09.08.2021 10:34.gpx'
    outfilename = "output/gpx1.png"
    drawGPX(master_title, filename, outfilename)

    master_title = "Electric Scooter Stock Firmware"
    filename = data/only_scooter/DarknessBot 09.08.2021 14:18.gpx'
    outfilename = "output/gpx2.png"
    drawGPX(master_title, filename, outfilename)
```

Listing C.2. draw_GPX.py

Appendix D

AutoCAD project of the wheelchair bracket

Below is shown the *AutoCAD* project drawing with the actual measurements calculated during the design phase of the bracket that must be mounted on the wheelchair to allow assembly between it and the electric scooter.

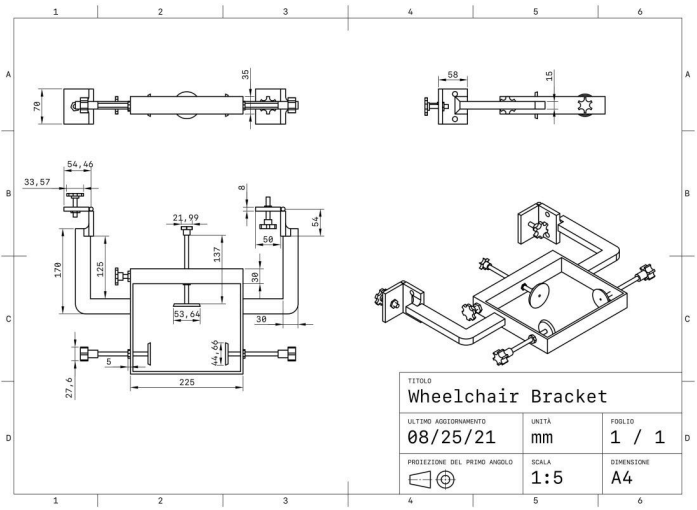


Figure D.1. 3D wheelchair bracket design

Bibliography

[1] Aldo Barillaro. *Che cos'è il firmware? E che differenza c'è tra BIOS e UEFI?* 2019. URL: https://www.informaticapertutti.com/che-cose-il-firmware-e-che-differenza-ce-tra-bios-e-uefi/#che-cose-il-firmware.

[2] Aletà, N. B., Alonso, C. M., Ruiz, R. M. A. *Smart mobility and smart environment in the Spanish cities - Transportation research procedia*. 2017. URL: https://www.sciencedirect.com/science/article/pii/S2352146517303654.

[3] arber, R., Klein, B. *UC Berkeley study finds electric scooter market divide*. 2019, February 14. URL: https://www.dailycal.org/2019/02/14/uc-berkeley-study-finds-electric-scooter-market-divide/.

[4] ARM DDI. *ARM®v7-M Architecture Reference Manual*. 2006. URL: https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/readings/ARMv7-M_ARM.pdf.

[5] Attify. *Hacking BLE*. 2019. URL: http://www.blog.attify.com/the-practical-guide-to-hacking-bluetooth-low-energy.

[6] Battarra, R., Gargiulo, C., Tremiterra, M. R., Zucaro, F. *Smart Mobility in Italian Metropolitan Cities: A comparative analysis through indicators and actions - Sustainable cities and society*. 2018. URL: https://www.researchgate.net/publication/325631788_Smart_Mobility_in_Italian_Metropolitan_Cities_A_comparative_analysis_through_indicators_and_actions.

[7] Batty, M., Axhausen, K. W., Giannotti, F., Pozdnoukhov, A., Bazzani, A., Wachowicz, M.,... *Smart cities of the future - The European Physical Journal Special Topics*. 2012. URL: https://link.springer.com/article/10.1140/epjst/e2012-01703-3.

[8] CBInsights. *Micro Mobility Markets and Companies*. 2020. URL: https://www.cbinsights.com/research/report/micromobility-revolution/.

[9] Chengdu PixelCyber Network Technology Co., Ltd. *Thor HTTP Sniffer/Capture*. 13 Jun 2020. URL: https://apps.apple.com/us/app/thor-http-sniffer-capture/id1210562295.

[10] Dedhia, Prashant. *The History of Electric Scooter*. 2019. URL: https://www.linkedin.com/pulse/history-electric-scooters-prashant-dedhia-negotiation-%20ninja-/.

[11] Dediu, Horace. *The Micromobility Definition - Micromobility Indutries*. 2019. URL: https://micromobility.io/blog/2019/2/23/the-micromobility-definition.

[12] Dediu, Horace. *The Micromobility Definition - Micromobility Indutries*. 2019. URL: https://micromobility.io/blog/2019/2/23/the-micromobility-definition.

[13] Dungz, Phan Hoang. *History | kickscooter*. 2020. URL: https://thegioitienganhvietnam.wordpress.com/history-about-kick-scooter/.

[14] Ebike portal. *Ogdeb Bolton Jr and his 1895 Hub Motor Ebike*. 2020. URL: http://www.ebikeportal.com/history/ogden-bolton-jr-and-his-1895-hub-motor-ebike.

[15] electric-scooter.guide. *The Leader in Real Electric Scooter Reviews*. 2021. URL: https://electric-scooter.guide.

[16] eMobility. *m365 Tools per m365/Pro/1S/Pro2, G30, ES and more*. 2018. URL: https://linux.die.net/man/1/hcitool.

[17] etransport. *ES2ESC*. 6 Jan 2019. URL: https://github.com/etransport/ninebot-docs/wiki/ES2ESC.

[18] *European Disability Forum*. 2021. URL: https://www.edf-feph.org.

[19] F0xMaster. ""Ninebot IAP guide - how to flash your scooter with a cable"". In: 2020. URL: https://www.scooterhacking.org/forum/viewtopic.php?t=252.

[20] Farrell, Richard. *Gustave Trouve Pioneers Electric Transport in 1881 - News about Energy Storage, Batteries, Climate Change and the Environment*. 2018. URL: https://www.upsbatterycenter.com/blog/gustave-trouve-electric-tricycle.

[21] Gerald Combs. *Wireshark*. 1998. URL: http://wireshark.org.

[22] ISTAT. *Audizione dell'Istat presso il Comitato Tecnico Scientifico dell'Osservatorio Nazionale sulla condizione delle persone con disabilità*. 2019. URL: https://www.istat.it/it/files/2021/03/Istat-Audizione-Osservatorio-Disabilit%C3%A0_24-marzo-2021.pdf.

[23] Jaya Gill, Jason Hu, Lucia Shan, Sam Thackray, Spencer Unger. *Over-the-Air Updates with Electric Vehicles*. 2020. URL: https://www.newmediabusinessblog.org/index.php/Over-the-Air_Updates_with_Electric_Vehicles.

[24] KToffel and Schorlescooter. *Scooter Companion*. 2021. URL: https://scootercompanion.github.io.

[25] Linden, John. *A History of Electric Scooters*. 2020. URL: https://www.carcovers.com/resources/a-history-of-electric-scooters/.

[26] Maxim Krasnyansky and Marcel Holtmann. *hcitool*. 2017. URL: https://linux.die.net/man/1/hcitool.

[27] Merriam-Webster. *Scooter | Definition of Scooter by Merriam-Webster*. 2020. URL: https://www.merriam-webster.com/dictionary/scooter.

[28] monopattinoitalia.it. *Miglior firmware Xiaomi V2*. 12 Nov 2020. URL: https://www.monopattinoitalia.it/miglior-firmware-xiaomi-v2/.

[29] National Observatory on Sharing Mobility. *Report: National Observatory on Sharing Mobility*. 2019. URL: http://osservatoriosharingmobility.it.

[30] Nobuhiro Iwamatsu. *gattool*. 2010. URL: http://manpages.ubuntu.com/manpages/cosmic/man1/gatttool.1.html.

[31] Nordic Semiconductor. *Nordic Semiconductor Infocenter*. 2021-07-30. URL: https://infocenter.nordicsemi.com/topic/struct_nrf51/struct/nrf51.html.

[32] NSA. *Ghidra*. August 6, 2021. URL: https://ghidra-sre.org.

[33] Papa, E., Lauwers. *Smart mobility: opportunity or threat to innovate places and cities. In 20th international conference on urban planning and regional development in the information society (REAL CORP 2015)*. 2015. URL: https://www.researchgate.net/publication/303179184_Smart_mobility_Opportunity_or_threat_to_innovate_places_and_cities.

[34] Patmont, Steven J. *Electric scooter. United States of America Patent US5775452A*. 2016. URL: https://patents.google.com/patent/US5775452A/en.

[35] Patrick Nohe. *Executing a Man-in-the-Middle Attack in just 15 Minutes*. 2021. URL: https://www.thesslstore.com/blog/man-in-the-middle-attack-2.

[36] quifinanza.it. *Leggi e normativa sul monopattino elettrico*. 2021. URL: https://quifinanza.it/green/leggi-e-normativa-sul-monopattino-elettrico/489864/.

[37] Rani Idan. *Don't Give Me a Brake – Xiaomi Scooter Hack Enables Dangerous Accelerations and Stops for Unsuspecting Riders*. Feb 12 2019. URL: https://blog.zimperium.com/dont-give-me-a-brake-xiaomi-scooter-hack-enables-dangerous-accelerations-and-stops-for-unsuspecting-riders/.

[38] Riccardo Bignucolo. *Un Evoluzione a Piccoli Passi*. 2018. URL: https://bici-e-bike.home.blog/2018/11/19/evoluzione-a-piccoli-passi/.

[39] ScooterHacking.org. *Ninebot scooters communication protocol*. 2021-04-11. URL: https://wiki.scooterhacking.org/doku.php?id=nbdocs.

[40] Simone 'evilsocket' Margaritelli. *::bettercap*. 2016. URL: https://www.bettercap.org.

[41] STMicroelectronics. *Mainstream Performance line, Arm Cortex-M3 MCU with 64 Kbytes of Flash memory, 72 MHz CPU, motor control, USB and CAN*. 2019. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html.

[42] STMicroelectronics. *STM32 ST-LINK utility*. 2019. URL: https://www.st.com/en/development-tools/stsw-link004.html.

[43] Wibree. URL: https://www.classicistranieri.com/it/articles/w/i/b/Wibree.html.

[44] Wikipedia contributors. *Bluetooth Low Energy — Wikipedia, The Free Encyclopedia*. 2018. URL: https://it.wikipedia.org/wiki/Bluetooth_Low_Energy.

[45] Wikipedia contributors. *Scooter-sharing system*. 2020. URL: https://en.wikipedia.org/wiki/Scooter-sharing_system.