POLITECNICO DI TORINO

Master of Science

in Electronic Engineering

Master Degree Thesis

Design of a smart sensor for obstacles detection to be installed on electric scooters or bicycles

Supervisor
Prof. Daniele Trinchero

Co-Supervisor
Ing. Alberto Bertone

Candidate
Mauro d'Addato

Academic Year 2020 - 2021

Acknowledgements

First of all, I would like to thank my business supervisor Alberto Bertone who allowed me to conclude my university path in Teoresi company with this beautiful thesis project and Professor Daniele Trinchero for accepting to be my supervisor.

Then, I would like to thank all the people who have been close to me over the years, allowing me to live a unique university experience. In particular, I would like to thank all the roommates (Vinc, Dome and Moha) who had to put up with me and my second family (Airasca and Ferrucci house) with whom I pleasantly spent all my university years.
I would like to thank all the people who, more or less, have become part of my life, giving me unforgettable experiences and allowing me to have both intellectual and character growth.

In particular, I would like to thank Eleonora, my sweet half, that, during this long journey, has always been close to me despite a thousand kilometers separated us. It wasn't easy but we always fought and resisted.

Finally,
there is someone who has always been by my side;
there is someone who has always believed in me, especially in most difficult moments;
there is someone who taught me to be brave, determined and never give up;
there is someone who taught me to be humble while never making me miss anything in this great and important period of my life…

…to my family,

my biggest strength,
my strongest support,
my priceless wealth,

always
and forever

Abstract

Nowadays, electricity is increasingly becoming the centre of sustainability and is revolutionizing the mobility of people. The future is electric.
Electric vehicles have widely introduced themselves into modern society and will undoubtedly be the protagonists of the new urban and extra-urban mobility. Everything evolves. The world is constantly developing to make life better and easier. Electric bicycles and scooters, day by day, are taking root more and more in the daily mobility of people by reconstructing the concept of light urban mobility.
Unfortunately, these vehicles, even if eco-sustainable, are not yet equipped with any system that allows drivers to recognize any dangers along the way that could jeopardize the safety of these vehicles. My thesis goes in this direction, with the aim of simplifying the way of getting around cities trying to make movements safer. Hence my thesis project, focused on the development of a smart sensor for obstacle detection. I immediately showed interest in the idea of designing a real-time sensor that could detect obstacles on the road and help the driver to avoid them. In particular, the creation of a sensor prototype and the development of the logic for its programming interested me a lot.
The purpose of this work is to understand if it is possible to design a sensor able to correctly detect objects or road irregularities and to communicate with the user in real-time. In collaboration with Teoresi company, I created a POC (proof of concept) to prove its feasibility.
The project started from scratch, so I had to take initiative on everything: make initial assumptions, choose and assemble the main and peripheral components, connect and design all the hardware, program the software by writing code and studying its logic, test the sensor under different conditions by analysing all the data and, finally, examine the results. Therefore, I had to take care of every single part of the project and this is the reason why, initially, I was a little bit scared, but then, I got very excited.

Contents

Introduction

In recent years, environmental problems due to CO_2 generated by urban traffic have led many cities to block entire categories of vehicles with emissions above the limits. This is the reason why many companies have begun to invest more in electric vehicles, allowing for a rapid and strong introduction of eco-sustainable light mobility, especially within large metropolitan cities.
In addition, due to the serious covid-19 pandemic, after the period of the first lockdown, in Italy for example, the Ministry of the Environment provided the "Mobility Bonus": a measure introduced with the aim of encouraging eco-sustainable private mobility.
Electric scooters, in particular, are among the protagonists of post-lockdown mobility. The convenience of these vehicles, their low costs and the possibility of social distancing have led to a boom in both purchases and the flowering of sharing services. [2] This choice, also linked to independence and fluidity in travel, has led many people, especially commuter workers, not to privilege either public transport such as subways and buses, or private vehicles and taxis (subject to traffic and parking).
But, on the other hand, with the increasing use of electric two-wheeled vehicles, the number of accidents involving these vehicles has also grown rapidly. The causes of these accidents are due to the dangers that, very often, the driver of the vehicle encounters during the journey travelled, starting from obstacles due to the nature of the carriageway up to moving obstacles such as pedestrians and motor vehicles that can represent a much more unpredictable danger. If we exclude the accidents caused by driver distraction, autonomous falls are the most recurrent cause, due to the vehicles' nature which, being very unstable, suffer from many road bumps such as holes and disconnections. [2]
Each year, the Central Statistics Service of Accident Insurance (SSAINF) records around 31,000 cases of bicycle accidents.

Suva, one of the leading companies for Swiss compulsory accident insurance, analyzed the accident reports and found that in about half of the cases the accident was almost always due to a specific danger.
According to statistics, the most frequent danger for cyclists is represented by sidewalks and cracks in the road surface. But even the tram tracks represent a considerable danger together with trees, shrubs, roots, escarpments or speed bollards. In addition to these, there are accidents caused by other motor vehicles, other cyclists or other people. In these cases, predictive driving is important to allow to identify in time the dangers and moves of other road users. [3]
It often happens, however, that cyclists do not realize the extent of the danger or can get distracted while driving. Therefore, the goal of the project is to create a sensor able to detect, in real time, the dangers that can put the driver at risk, becoming the cause of an accident.
To verify the feasibility of our idea, we will create a POC (proof of concept). The problem will be analyzed and studied from all points of view, a prototype of the sensor will be created and, at the end, the necessary tests to verify its operation will be performed.

Chapter 1

Preliminary Sensor Characterization

The idea of the project was born for my thesis purposes, therefore, I had to start from the scratch from all points of view. Before proceeding to search for the sensor, I analyzed and chose the initial conditions that could best characterize it. To draw up the guidelines useful for sensor design, I initially studied all the key points of the project to be developed and the characteristics necessary to define it, including:

- Use period
- Sensor position
- Pointing Distance
- Type of detected obstacles

1.1 Use period

Electric scooters and bicycles are vehicles that can be used at any time, both day and night, and throughout the year (although the hottest periods are preferred). Furthermore, the climatic conditions represent a very important factor that could greatly influence the measurements of a sensor and, despite the low percentage of use of these means in unfavorable conditions (such as fog, rain or snow), it must, however, be taken into consideration. and I need to analyze the behavior of the sensor in such situations, since it is precisely in these climatic conditions that the risks and dangers on the road increase. Therefore, there is a need for a sensor capable of being effective all day in favorable climatic conditions, and possibly, even in unfavorable ones.

1.2 Sensor position

The dangers that the driver may encounter while driving may arise in front of him (under his eyes) and, sometimes, even behind him.
The rear-end collision by cars or other motor vehicles, in addition to being very dangerous (if not lethal), due to the great disparity in mass between the two vehicles, is also unpredictable, since the danger presents itself behind the driver. The choice of the sensor position becomes, therefore, relevant to discriminate the type of obstacles/dangers that may arise.

1.2.1 Front/Back choice

The sensor project is aimed at both types of electric vehicles, and must therefore be able to be used on both scooters and electric bicycles.
Despite the similarity between the dangers to which the two electric vehicles are exposed, there are, however, some structural differences that distinguish them.
In the electric scooter, unfortunately, the rear is almost inexistent, since it is represented by the rear wheel alone plus the fender, if any. The positioning of a sensor at a few centimeters from the roadway would therefore become uncomfortable and ineffective, as well as more easily breakable.
Opposite considerations must be done for the front of the two vehicles which, on the other hand, are somewhat similar due to the presence of the handlebar, at a height of about 1 meter. A large number of these electric vehicles already have a small display on it that indicates the speed or the state of charge of the battery and so, it could also house the sensor.

Fig. 1.2.1 : Representative pics of the two vehicles

1.3 Type of detected obstacles

The obstacles that the driver may encounter along the roadway are not few. Predictive guidance is important for anticipating dangers that can suddenly arise. Many accidents are due to unevenness in road surface which can manifest in many ways:

- Shrubs e roots
- Tram tracks
- Ditches
- Road Bumps
- Sidewalks and steps

Others, instead, are due to moving obstacles such as animals, people or other vehicles which, being less predictable, can be the most dangerous. [3]
The choice made for this project was to start from the recognition of obstacles of different materials and sizes, proceeding, then, towards a discrimination between the most frequent obstacles in a usual urban ride, such as ditches, bumps, steps/sidewalks and obstacles of larger size.

1.4 Pointing distance

Distance range is a fundamental feature in a measurement sensor and is one of the key factors in sensor design.

“But… why are we talking about pointing distance and not about maximum distance range?”

To detect the different obstacles, in fact, the key idea was to tilt the sensor down, the way to cross the roadway at a precise distance. In this way, we obtain a constant average distance between the sensor and the ground, which we call pointing distance, and on whose variation the whole logic of the project is concentrated.

1.4.1 Speed and Stopping distance

In order to correctly prevent obstacles and, thus, calculate the pointing distance, the speed and stopping distance of the vehicles must be taken into account.
The law of 28 February 2020 established the equation of electric scooters with cycles, placing constraints on the continuous nominal power of the vehicle (not exceeding 500 W) and for the maximum speed that cannot exceed 25 km/h. [4]
Therefore, taking this speed as the maximum limit, we calculated the maximum stopping distance, to understand how soon the obstacle must be detected so that the driver has the sufficient time to dodge or stop. The stopping distance is calculated with the following formula:

$$stopping\ ditance = reaction\ space + braking\ distance$$

The reaction time is 1 second but can vary, depending on the driver's clarity and his reflexes.
The braking distance is the distance traveled by a vehicle from the moment the braking action begins until it stops completely and depends on many factors, including:

- Speed
- Deceleration
- Friction coefficient between tires and road
- Vehicle mass
- Road slope

There are many factors to consider, but, for simplicity, the following formula has been used:

$$braking\ distance = \frac{v^2}{2 \cdot g \cdot \mu} = \frac{(6{,}94\ m/s)^2}{2 \cdot 9{,}8\ m/s^2 \cdot 0{,}8} = 3{,}07\ m$$

where, the earth gravity acceleration value 'g' is used as deceleration under braking and the friction coefficient 'μ' is equal to the optimal asphalt condition value, while 'v' is the vehicle speed. It should be noted that the friction coefficient can be reduced from 0.8 to a value of 0.05 in icy road conditions, thus, making the braking distance 16 times greater. [5] Taking into account the reaction space at maximum speed and at braking time t equal to 1 second, we obtain:

$$reaction\ space = 25\ km/h \cdot 1\ s = 6{,}94\ m$$

$$stopping\ distance = 6{,}94\ m + 3{,}07\ m = 10{,}01\ m$$

The stopping space also represents the minimum distance in which an object must be found in order to be avoided. The sensor, therefore, will be positioned so that the beam points the road at a distance of 10 m, as reported in the calculations just made.

1.4.2 Height e inclination

The height and the inclination of the sensor both depend on the pointing distance. The best position for the sensor, of course, will be the highest possible because this allows to obtain the pointing distance with the greatest angle of inclination. If, for example, you think of positioning the sensor at the bottom, the angle of inclination is reduced so much that the direction of the sensor becomes almost parallel to the ground. If you take into account the oscillations of the vehicle while driving, you immediately understand that this brings to have a high number of errors in the measurements.

The graph (indicative) in Fig. 1.4.2 shows how, as the angle of inclination α decreases, the pointing distance *d* becomes ever closer to the value of the stopping space.

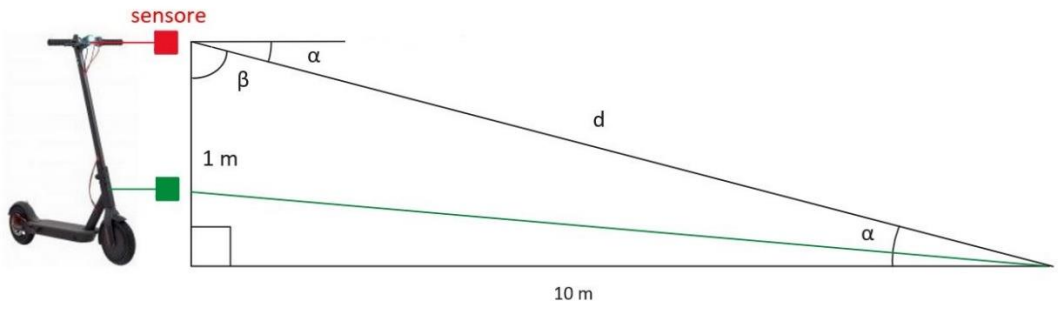


Fig. 1.4.2 : Difference between top (red) and bottom (green) sensor placement

It was, therefore, chosen to position the sensor on the handlebar, where the maximum height is reached for both vehicles.

Once the sensor height has been established, which on average is about 1 m (based on the height of the handlebar) and the maximum stopping distance just computed, it is finally possible to calculate the pointing distance '*d*' and the inclination of the sensor '*α*' by applying trivially the theorems of right triangles.

$$d = \sqrt{(10\,m)^2 + (1\,m)^2} = 10{,}04$$

$$\frac{1\,m}{\sin(\alpha)} = \frac{d}{\sin(90°)} \quad \rightarrow \quad \alpha = \sin^{-1}\left(\frac{1\,m \cdot \sin(90°)}{d}\right) = 5{,}68°$$

The value of *α* is obviously a theoretical and ideal value which in reality is neither feasible nor repeatable. In the test phase, further on, we will try to get as close as possible to this ideal value. It should also be noted that although the sensor has been positioned at the maximum height, the pointing distance and the stopping space differ by only 4 cm. This means that the sensor could be subject to oscillations that cause non-negligible measurement variations. This structural constraint could be a problem for sensor operation. Later it will be explained how the problem was partially avoided via software with an algorithm that filters the data received from the sensor.

Chapter 2

Sensor search

The number of available sensors on the market is increasing rapidly, due to the great competition that has now been created in the IoT field. With the presence of a large number of sensors on the market, it becomes essential to dedicate time to this choice, as it allows us to select the most appropriate and useful features for the final purpose. The search for the distance sensor to be used for obstacle detection is one of the most important phases of the project, since all the study that will be done during the thesis, derives from the sensor data.

2.1 Distance sensor technology

Distance sensors can solve an incredible variety of problems and, depending on the application, it is necessary to select the distance sensor with the most suitable measurement technology.
The principle of operation of distance sensors is very similar, but differs in the construction method and in the technology used. The distance of the detected obstacle depends on the variation between the emission of a signal and its return. This variation can be measured either through the time of the return signal or through the intensity of the return signal.
After extensive research, the technologies on the market turned out to be numerous. Excluding proximity sensors and all technologies with a distance range of less than one meter, the main remaining ones are:

- Ultrasonic Sensor
- Infrared Sensor
- LiDAR Sensor

Radar sensors were excluded from the list, since, a priori, they were considered unsuitable for our needs.
Radar signals work better with obstacles that are far away from the receiver (much more than 10 meters). Their waves have dimensions that don't go below the millimeter allowing them to be effective in any climatic condition but making it difficult to discriminate obstacles (which is our purpose), especially if they are in

motion. In fact, although the millimeter can be considered a relatively small measure in everyday life, in reality in the spectrum of electromagnetic waves it is one of the largest wavelengths, if we consider that the wavelength of UV rays is 100,000 times smaller. Finally, we also need to take into account the cost of a radar sensor that is relatively high.

2.1.1 Ultrasonic sensor

The ultrasonic sensor, also known as “Sonar”, detects the objects distance by emitting high frequency ultrasonic waves (>20 kHz) that are inaudible to the human ear (which perceives sounds between 20 Hz and 20 kHz).
Depending on the type of sensor used, distances can vary, from a few centimeters up to tens of meters.
Ultrasonic sensors emit ultrasonic pulses that travel in a cone-shaped beam using a vibrating device known as a piezoelectric transducer, which generates the ultrasonic wave. [6]

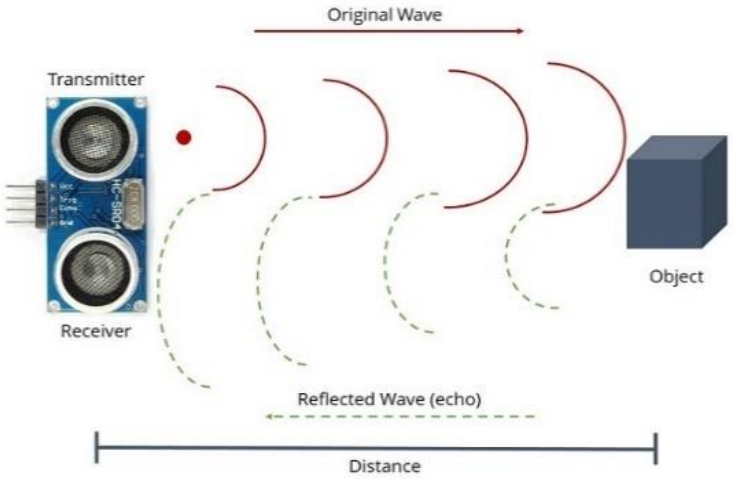


Fig. 2.1.1 : Ultrasonic sensor

The ultrasonic pulses are reflected by an object and the generated echo is received by the ultrasonic sensor and converted into an electrical signal by the piezoelectric transducer. This principle is known as sound propagation time. The sensor measures the interval between the ultrasonic pulse emitted and the echo received and calculates the distance to the object using the speed of sound in the air, which at room temperature is approximately 344 m/s.
Ultrasonic sensors are able to detect objects of different material regardless of their shape, color or state even if some objects could limit the ultrasonic sensors action

range; for example, objects with a large, smooth and sloping surface, or objects made of porous materials.

In addition to the surface properties, the sensing distance of an ultrasonic sensor also depends on the object angle. The longest sensing distances are achieved with objects that have a flat surface (standard reflector) placed at an exact right angle (90°) to the sensor axis. Very small objects or objects that reflect sound only partially reduce the detection distance. [7]

2.1.2 Infrared sensor

The infrared sensor detects the distance by emitting an IR ray, invisible to the human eye, and calculating the angle of reflection of the ray. There are different types of infrared transmitters based on their wavelengths, output power and response time.
The IR sensors have two lenses: an IR LED emitter lens that emits a light beam and a photodiode that is sensitive to the position on which the reflected beam will fall. IR distance sensors work according to the triangulation principle; they measure the distance based on the angle of the reflected ray.

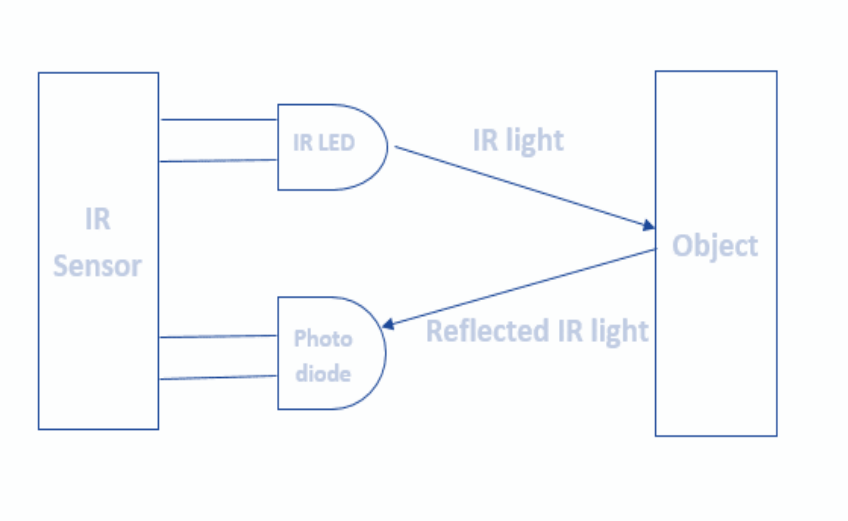


Fig. 2.1.2 : Infrared sensor

The infrared light emitted by the IR LED emitter generates a light beam that hits the object and is reflected from a certain angle. The reflected light, then, reaches the photodetector which determines the distance of the reflecting object. [8]
The photodiode, in fact, is sensitive to the light emitted by the LED. The resistance of the photodiode and the output voltage change in proportion to the received light; therefore, the sensor defines the distance based on the intensity of the received signal. [9] These sensors are efficient both day and night even if some sensors can be

affected by sunlight, especially if it is directed against the photodiode. Common infrared sensors tend to have small dimensions and are able to measure the distance of objects that have complex surfaces, but, although very accurate, they have not a very high reading frequency. Also, the action range of infrared sensors is only a few meters (usually <5m). [8]

2.1.3 LiDAR sensor

The acronym LiDAR (Light Detection and Ranging) identifies the technology that measures the distance from an object by illuminating it with a laser light and which, at the same time, is able to return high-resolution three-dimensional information on the surrounding environment. The LiDAR sensor can be considered a laser distance sensor and measures the distance of obstacles by emitting light waves (instead of sound or radio waves). The distance to the object can be determined by measuring the elapsed time between the emission of the pulse and the reception of the backscattered signal.

Knowing that the propagation speed of light is fixed (c ≈ 300,000 km / s), it's possible to easily calculate the time it takes for a light beam to go from a source to a (reflective) target and to go back towards the light detector (placed next to the emitting light source).

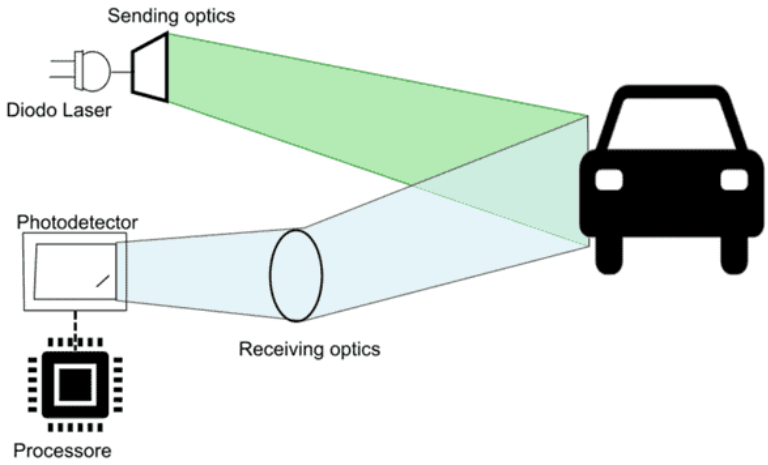


Fig. 2.1.3: LiDAR sensor

This measurement principle is usually referred to as "Time of Flight" (ToF). The time of flight can be obtained by sending an impulsive signal using a laser, but also by measuring the phase and frequency of the reflected light signal with respect to a reference signal. [10]

LiDAR sensors have a very high resolution. In fact, light has wavelengths λ shorter than those of radio waves, infrared or ultrasound waves and this increases the

detection resolution allowing it to recognize objects of any shape and size. Furthermore, the LiDAR reading frequency is very high and guarantees rapidity in measuring distances.
The sensor sends and receives laser pulses in nanoseconds, obtaining a high number of measurements per second. This has considerable importance as it allows to detect even moving obstacles, where quick measurements are needed. [8]

2.1.4 Technology comparison and choice

Analyzing and choosing the most suitable technology is essential to reach the right compromise between the advantages and limitations that it can involve, since it allows the designers the best choice for their application. Key features are listed below:

- Distance range
- Resolution
- Reading Frequency
- Environmental conditions dependency
- Obstacle complexity dependency
- Costs

The ultrasonic sensor is not affected by the color and transparency of the object and works well even in dark places as it uses sound waves, but has a low reading frequency and, above all, a very limited range. The same considerations can be made for the infrared sensor that can detect objects with complex surfaces, even in the dark, but has a range limited to only few meters away.
Both sensors, ultrasonic and infrared, have a lower reading frequency than the LiDAR sensor since their principle of operation is based on waves with a much lower λ wavelength, thus allowing to carry out much less measurements per second.
The LiDAR sensor, therefore, is the most complete and performing as it has a very high range (some even reach more than 1000 m) and a high reading frequency, such as to allow the detection of objects that move very quickly, as anticipated in the previous paragraph. [8]
Furthermore, it is necessary to take into consideration the study on the sensor positioning carried out in the previous chapter where we found the need to place the sensor at the maximum vehicle height (1m), thus making it possible to reach a pointing distance of 10 m only with an inclination angle α not exceeding 6°.
Such a small inclination angle greatly affects the efficiency of a distance sensor, making it more difficult to obtain correct measurements. It is especially in this case that a higher reading frequency, such as that of LiDAR sensors, becomes fundamental, since a greater number of readings leads to a greater quantity of non-erroneous measurements.

	Ultrasonic	Infrared	LiDAR
Suitability for Long Range Sensing	No	No	Yes
Resolution	High	Variable according to distance	Very High
High reading frequency	No	No	Yes
Sensitive to external conditions	Yes	No	No
Suitability to use for complex objects	No	Yes	Yes
Costs	Low	Low	High

Tab. 2.1.4: Comparison of technologies

As can be seen from the table (Tab. 2.1.4), LiDAR technology is the best from almost all points of view. What makes difference is, above all, the high resolution and the speed of measurement, even at long distances, which allow to discriminate many more objects than other technologies, whatever their size or shape.
LiDAR, however, is very expensive and this represents, as mentioned before, the only concrete drawback of this technology. On the other hand, this project does not have as its purpose the search for an inexpensive sensor but has the single goal of verifying the feasibility of a sensor effective for detecting obstacles. Despite its cost, therefore, LiDAR will be the technology on which we will lay the foundations of the whole project.

2.2 LiDAR sensors

LiDAR technology, acronym for Light Detection and Ranging or, sometimes, expressed as Laser Imaging Detection and Ranging, is a technique that today, is used in everything related to distance measurement, as it allows you to measure, in an extraordinarily fast and precise manner, the distance between the measuring point and the destination point.
These sensors are often used as scanners: they can easily "read" the territory by identifying its hills, ditches, their depth and so on; for this reason, in recent years this technology has been increasingly used in mapping the morphologies of the territory, helping to establish the level differences between the terrains. With the data provided by the LiDAR it is possible to create numerous Digital Surface Models (DSM) with a very high precision and a very small margin of error (it is not a coincidence that this margin is often even measured in a few centimeters).
The laser scanning technique allows you to reproduce the environment in an extremely detailed manner. For this reason, today it is one of the most widely used localization and navigation systems, so much so that people make more and more extensive use of it and its applications are greatly increasing. [11]
Even the automotive sector, therefore, has undergone a profound innovation for some years thanks to this new technology that has become fundamental in the development of ADAS systems (acronym for Advanced Driver Assistance Systems): advanced systems that assist vehicles making them automated and safe. The LiDAR sensor, along with other complementary sensors, helps achieve complete automation as it provides highly accurate data. The accuracy of the measurements and the high range of this sensor allow it to detect objects very quickly, therefore, it is able to provide help to the driver in real time. [12]
Unlike the automotive field, no particular driver assistance systems have yet been developed in two-wheeled vehicles. The LiDAR sensor could provide the driver with quick information on obstacles along the way helping him to avoid them.
It can certainly be admitted that this technique is one of the most innovative and useful ever. More and more people are turning their attention to LiDAR technology, even if it is still a technology that has the drawback of not being cheap. [11]
But many companies are already trying to cut the its production costs…

2.3 Sensor choice

After considering and analyzing all available sensor technologies and identifying LiDAR as the best, we moved on to a market research.
Although it is still an emerging technology, a fair amount of LiDAR sensors is already available on the market and it was necessary to choose the one that represented the best compromise in terms of characteristics, in particular, distance range, cost and performance.
After careful research, the characteristics of various sensors on the market were evaluated and the final choice fell on the "LIDAR-Lite v3".
The choice was heavily influenced by the ability to customize and configure the sensor through parameters, used for the distance measurement algorithm, including range, speed and sensitivity. Furthermore, since it is one of the most commercialized LiDAR sensors, lots of information is available online and this is, undoubtedly, very relevant for a quick and easy implementation and integration of the device.

Fig. 2.3.1 : LIDAR-Lite v3

2.3.1 LIDAR-Lite v3

The LIDAR-Lite v3 (Fig. 2.3.1) is a compact and high-performance optical distance measurement sensor and uses a single-chip signal processing method with minimal hardware. This sensor is highly configurable by the user so you can adjust between accuracy, operating range and measurement time. It can be used as a building block for applications where small size, light weight, low power consumption and high performance are important factors, such as robots and drones. [13][14]

In the LIDAR-Lite v3, a laser beam is emitted from the transmitter and the time it takes for reflection to go back is measured by the receiver. The distance can be calculated using the following equation:

$$d = \frac{c \cdot t}{2}$$

where, '*d*' is the distance of the reflected object in meters, '*c*' is the speed of light (3.0 x 10^8 m/s) and '*t*' is the time it takes (in seconds) for the light to leave the transmitter, bounce off the object and return to the receiver. Finally, it divides by 2 since the light must make a double trip (towards the object and back). [10]

The unique signal processing method transmits an encoded signature and searches for that signature in the return signal, enabling highly reflective detection with eye-safe laser power levels.

Before performing the first measurement, the LIDAR-Lite v3 performs a correction, or "bias correction", allowing you to refine the accuracy of the measurements that may vary based on changes in ambient light. This allows for maximum sensitivity. The device thus sends the signal directly from the transmitter to the receiver, setting the delay for a reference distance and periodically recalculating this delay after several measurements.

After making the correction, the device performs a series of acquisitions to obtain a measurement. It should be borne in mind that the LIDAR-Lite v3 has a very high reading rate and is able to provide a large number of measurements in a few seconds (including bias correction). [15]

In the following chapters we will analyze all the hardware and software features related to the LiDAR and the rest of the components used to complete the sensor project.

Chapter 3

Hardware Design

The purpose of this project phase is to create the sensor prototype that will be installed on an electric scooter or bicycle to study and test its operation. The creation of a prototype allows the designer to better analyze the interaction between hardware and software and to make all the necessary changes to correctly study the operation of the device and correct any problems that may arise. After having found the right measurement sensor and having characterized it at its best, I analyzed all the physical and electrical characteristics of the sensor and the communication protocol that allows the LIDAR-Lite v3 to communicate with the microcontroller and consequently with the user. In particular, the prototype created is composed of several devices including:

- LIDAR-Lite v3 sensor
- Elegoo Uno R3 microcontroller
- Display LCD
- Led RGB
- Passive Buzzer
- SD card reader
- Power bank
- Other passive elements

3.1 LIDAR-Lite v3 characteristics

The LIDAR-Lite v3 sensor is very light and compact. Its weight, of only 22 g, and its small size allow it to be used in many applications (Tab 3.1a). The device is equipped with a 905 nm (1.3 W) single-stripe laser transmitter, with an optical aperture of 12.5 mm (Tab 3.1b).
The diffusion of the laser beam depends on the distance of the obstacle: for very small distances (<1m) the diameter of the beam is as large as the size of the lens while, for distances greater than one meter it can be estimated simply using the following equation:

$$\frac{distance}{100} = laser\ beam\ diameter\ at\ that\ distance$$

The spread of the beam is about 8 mrad (~ 0.5 °). So, for example, at a distance of 10 meters, as in our case, the laser beam will have a diameter of 10 centimeters. [14]

Regulatory Approvals: *CLASS 1 LASER PRODUCT CLASSIFIED EN/EIC 60825-1 2014. This product is in conformity with performance standards for laser products under 21 CFR 1040, except with respect to those characteristics authorized by Variance Number FDA-2016-V-2943 effective September 27, 2016.* [15]

In short, as regards the safety of the laser beam, Class 1 indicates its safety under all conditions of common use.

Specifications	**Measurement**
Size	20 x 48 x 40 mm
Weight	22 g
Operating Temperature	-20 a 60 °C

Tab 3.1a: Physical Property

Specifications	**Measurement**
Wavelenght	905 nm (nominal)
Total laser power	1.3 W
Beam diameter at laser aperture	12 x 2 mm
Divergence	8 milliradian

Tab 3.1b: Laser Property

The Lite-v3 requires a voltage between 4.5 and 5.5 V DC (5 V nominal) with a power consumption of 105 mA during an acquisition which becomes 135 mA in continuous operation (Tab 3.1c). Therefore, it shows a very low energy consumption.

Specifications	**Measurement**
Power	5 Vdc nominal 4.5 Vdc min, 5.5 Vdc max.
Current	105 mA idle 135 mA continuous operations

Tab 3.1c: Electrical Property

The sensor has a laser beam that reaches 40m with an accuracy of ± 10cm, which is reduced to 2.5cm for distances of less than 5m.
The update rate of the LIDAR-Lite v3, in normal operation, is typically 270 Hz, but if it is used in "fast mode" it reaches 650 Hz and, for small distances, it also reaches an update rate greater than 1000 Hz, that is 1000 acquisitions per second. [15]
Since a measurement, by default, is processed over 5 acquisitions, the sensor provides a large amount of measurements in a short time. The number of measurements per second will be estimated later in the debugging phase.

Specifications	**Measurement**
Range (70% reflective target)	40 m
Resolution	± 1 cm
Accuracy < 5 m	± 2.5 cm typical*
Accuracy ≥ 5 m	± 10 cm typical
Update Rate (70% Reflective target)	270 Hz typical 650 Hz fast mode** > 1000 Hz short range only
Repetition rate	∼50 Hz default 500 Hz max

* Non linearità presente sotto 1 m
** Sensibilità ridotta

Tab 3.1d: Performance

3.1.1 Connection and wiring

The sensor can be connected to the microcontroller via a rectangular electric latch-lock port that connects the sensor to a 6-wire cable (Tab 3.1.1).
The wires have different colors in order to be more easily identified within the circuit: in particular, red and black are used for power supply, as is commonly the case, while green and blue are used for the I^2C connection.
Instead, the other 2 wires, orange and yellow, respectively represent the enabling of the power supply (via internal pull-up) and the control of the connection mode. [14]

Wire Color	Function
Red	5 V DC (+)
Orange	Power enable
Yellow	Mode control
Green	I²C SCL
Blu	I²C SDA
Black	Ground (-)

Tab 3.1.1: Connections

As previously anticipated, there are two configurations for this device:

- I²C – serial communication bus between device and microcontroller
- PWM – bidirectional signal transfer method that sends acquisitions and receives the distance measurement via the Mode Control pin.

According to the chosen communication mode, there is a different hardware configuration of the device, in order to optimize the circuit at an electrical level.
In this project, we will use the I²C serial interface.

3.1.2 I²C interface

The LiDAR Lite v3 sensor can be connected to an I²C bus as a slave, under the control of an I²C master device. Supports fast data transmission (fast mode) at 400 kHz. The I²C bus operates internally at 3.3 Vdc. An internal level change allows the bus to operate at a maximum of 5 Vdc. The internal 3 kΩ pull-up resistors ensure this functionality and allow the I²C host a simple connection.
The device has a 7-bit address (128 possible different addresses) with a default value of 0x62. The effective 8-bit I²C address is 0xC4 for writing and 0xC5 for reading. [15]

I²C protocol

The I²C protocol (Fig. 3.1.2) uses a bus with a clock signal (SCL) and a data line (SDA) and 7 possible addressing bits. A bus has two types of nodes:

- Master – the device that outputs the clock signal
- Slave – the node that synchronizes on the clock signal without being able to control it

You can have multiple slaves and multiple masters connected, but in our case, we only have one on each side. The master and the slave will be, respectively, the microcontroller and the LIDAR-Lite v3.

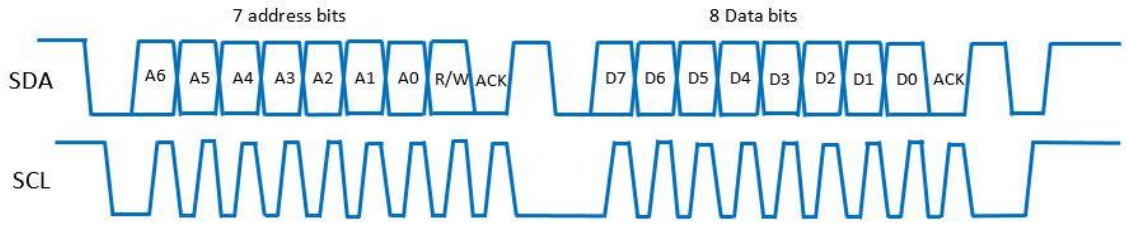


Fig. 3.1.2: I²C serial protocol

In the following points there is a brief description that explains how the I²C protocol works and how reading and writing occurs in a communication.

1. The master initiates the data transfer by establishing a start configuration, that is, when a high-to-low transition occurs on the SDA line while SCL = 1. The next byte is the address byte, which consists of 7 bits of address followed by the read/write bit (1/0) with '0' status indicating a write request. The write operation is used as the initial stage in both read and write operations.

2. The data is transmitted on the bus in a 9-bit sequence (8 data bits plus an acknowledgment bit). The transaction on the SDA line must occur when SCL=0 and remain stable when SCL = 1.

3. The data byte follows the address byte and is saved in the I²C control register with the address of the first register to be read (bit 5:0) together with the flags for the automatic increase of the register (bit 7).

4. If a read operation is required, a "stop bit" is inserted by the master at the end of the first data frame followed by the initialization of a new start configuration, i.e. the address of the slave with the "read bit" set to 1. The new address is followed by the reading of one or more bytes of data.
   After the slave has received confirmation of the address validity, the master releases the SDA line. At the end of the data reception, the master sends an "acknowledge bit" before continuing the cycle.

5. To continue with a write operation, Step 3 is followed by one or more 8-bit blocks with the slave sending an "acknowledge bit" at the end of each successful transfer. At the end of the write cycle, the master performs a stop configuration to end the operation. [15]

3.2 Elegoo Uno board

ELEGOO UNO R3 (Fig. 3.1.2) is a microcontroller based on the ATmega328. It has 14 digital input / output pins (6 of which can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header and a reset push button.

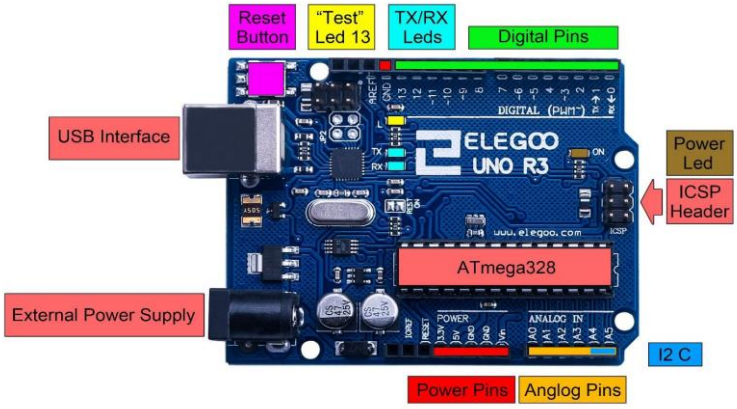


Fig. 3.2: Elegoo UNO R3 Board

Contains everything needed to support the microcontroller; it is sufficient to connect it to a computer with a USB cable or power it with an AC-DC adapter or with a battery. The UNO card differs from all previous cards in that it does not use the USB to serial FTDI driver chip. Instead, it has the Atmega8U2 programmed as a USB to serial converter.
The UNO version (1.0) and version 1.1 are the reference versions of Arduino. The Uno is the latest in a series of Arduino USB boards and is the reference model for the Arduino platform. [16]

3.2.1 Technical specifications

Power Supply
Elegoo UNO can be powered via USB connection or with an external power supply. The power source is automatically selected.
In our project, the microcontroller was powered in both ways, depending on the need. During the programming of the microcontroller, it was powered via USB connection while in the test phase a power bank was used to facilitate the tests on the vehicle.
The board can operate on an external 6 to 20V power supply. However, if powered with less than 7 V, the 5 V pin may supply less than 5 V and the board may be unstable. If more than 12 V are used, the voltage regulator may overheat and damage the board. The recommended range is, therefore, between 7 and 12 V.

The power pins are as follows:

- **VIN**. The input voltage to the Arduino board when using an external power source (as opposed to the 5V provided by the USB connection).

- **5V.** The regulated power supply used to power the microcontroller and other components on the board. It can be supplied by USB or another regulated 5V power supply.

- **3V3**. A 3.3 V power supply generated by the on-board controller. The maximum current draw is 50 mA.

- **GND**. Ground pin

- **IOREF**. This pin on the Elegoo board provides the voltage reference with which the microcontroller operates. [16]

Specifications	**Measurement**
Operating Voltage	5 V
Input Voltage (recommended)	7-12 V
Input Voltage (limit)	6-20 V
Digital I/O Pins	14
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current for I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
E^2PROM	1 KB (ATmega328)
Clock Speed	16 MHz
LED_BUILTIN	13
Dimensions	68.6 x 53.4 mm
Weight	25 g

Tab. 3.2.1: MCU Tecnical Specifications

Memory
The Atmega328 has a 32 KB flash memory for storing the code (of which 0.5 KB is used for the boot loader); It also has a 2 KB SRAM and a 1KB E^2PROM (which can be read and written with the E^2PROM library). [16]

Input/Output
Each of the 14 digital pins on UNO R3 can be used as input or output, using the "pinMode ()", "digitalWrite ()" and "digitalRead ()" functions. The pins are powered at 5 V. Each pin can supply or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kΩ. In addition, some pins have specialized functions:

- **Serial**: Pin 0 (RX) and Pin 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data (transistor-transistor logic).

- **External Interrupt**: Pin 2 and Pin 3. These pins can be configured to trigger an interrupt on a low level, a rising or falling edge, or a change in value.

- **PWM**: Pin 3, 5, 6, 9, 10 e 11. Provides an 8-bit PWM output with the "analogWrite ()" function.

- **SPI**: Pin 10 (SS), Pin 11 (MOSI), Pin 12 (MISO) and Pin 13 (SCK). These pins support the SPI communication (Serial Peripheral Interconnection).

- **LED**: Pin 13. There is a built-in LED connected to digital pin 13. When the pin is at the High value, the LED is on, when the pin is at the Low value, it is off.

The UNO board has 6 analog inputs, each of which provides 10 bits of resolution (i.e., 1024 different values). By default, they measure from GND up to 5 volts, although it is possible to change the upper limit of their range using the AREF pin and the "analogReference ()" function. Furthermore, the analog pins 4 (SDA) and 5 (SCL) support a second I^2C connection through the "Wire" library.
Finally, there are two more pins on the board:

- **AREF**. Reference voltage for the analog inputs.

- **Reset**. A Low value on this line resets the microcontroller.
  Typically used to add a reset button to the shields blocking the manual one on the board. [16]

3.2.2 Communication between Elegoo UNO and LIDAR -Lite v3

The figure below shows how the LiDAR sensor was connected to the Elegoo microcontroller. There are 4 connections, in particular:

- The red wire of the LIDAR-Lite (5 Vdc) has been connected to the 5V pin of the microcontroller.
- The green wire of the LIDAR-Lite (I^2C SCL) has been connected to the SCL pin of the microcontroller.
- The blue wire of the LIDAR-Lite (I^2C SDA) has been connected to the SDA pin of the microcontroller.
- The black wire (Ground) of the LIDAR-Lite has been connected to the GND pin of the microcontroller.
- The yellow and orange wires of the LIDAR-Lite have not been used as they are not needed for the I^2C connection.

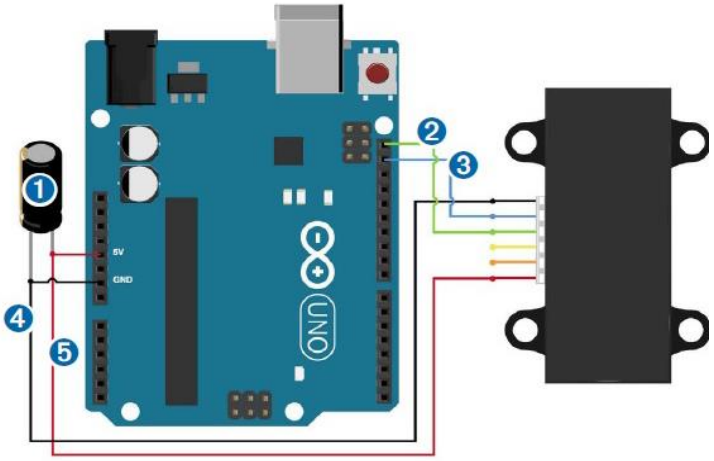


Fig. 3.2.2: Elegoo UNO - LIDAR-Lite connections

To maintain a constant voltage, it is recommended to insert a 680 µF capacitor between the power supply (5 V) and GND, as it mitigates the peak current at power up. [14]
Better if inserted as close as possible to the LIDAR sensor, as can be seen in Fig. 3.2.2. The recommended value is 680 µF, but any value that comes close is fine. In the project, for example, a 470 µF electrolytic capacitor was used (exactly the value that precedes the 680 µF one in the E6 series).

Note: care must be taken to correctly connect the capacitor since it is a polarized component and the inversion of polarity leads to a rapid destruction of the same.

3.3 Other components

The LIDAR-Lite v3 and the Elegoo UNO microcontroller are not the only components of the sensor prototype we are designing. In fact, other electronic components have been added to the circuit in order to facilitate both the debug part in the programming and the test part, immediately after, to get immediate feedback on the operation of the device.
The programming of the software requires a continuous checking procedure due to errors (bugs) that the programmer encounters frequently, both in the programming phase itself and in the following ones. The debugging activity is, therefore, very important, if not essential. The distance sensor, however, is a device that must be installed on a two-wheeled vehicle and it follows the need to have on-board components that allow to recognize system errors even without the use of a computer, especially in test phase.
In these cases, a visual/auditory feedback greatly helps the programmer to recognize the problems that may arise from incorrect writing of the code. In addition to being a quick debugging solution, acoustic and visual devices could also be very important to alert the user about the dangers that arise while driving.
To complete the circuit, we added an LCD display and an RGB LED for visual feedback and a buzzer for acoustic signaling. It should be underlined that the HMI present in the project is used only for the purpose of debugging and testing the device. The creation and optimization of the user interface is not part of the thesis project purpose.
To simplify the connection of all components, an EIC-104 1680-point Breadboard was also used, so as to have a base on which to compact the entire circuit.

3.3.1 LCD Display

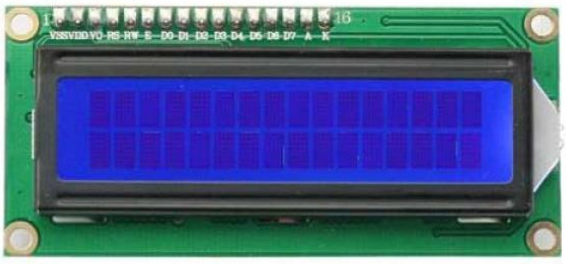


Fig. 3.3.1: LCD Display

The LCD display is a very useful component both in the debug phase and in the test phase to read the measurements made in real-time by the sensor. It features LED backlighting and can display two lines with up to 16 characters on each line. You can see the rectangles for each character on the display and the pixels that make up each character.

The display has a total of 16 pins to connect it to the board:

- **VSS**. Pin used to connect to GND.
- **VDD**. Pin used to connect to 5 V.
- **VO**. Pin used to adjust the LCD contrast.
- **RS**. Register select pin to manage where data is to be written in the LCD memory.
- **R/W**. Pin for selecting the writing or reading mode.
- **E**. Enable pin.
- **D0-D7**. Pin used to read and write data. Only 4 of the 8 pins are needed.
- **A, K**. Pins used to control the LED backlight. They are connected to 5 V and GND respectively.

The LCD display needs 6 Arduino digital pins, all set as digital outputs, to which RS, E, D4, D5, D6 and D7 pins will be connected. To be powered it needs 5V, therefore, we simply need to connect it to the breadboard power supply.
It is necessary to make a series of connections to complete the circuit that is shown in Figure 3.3.1.

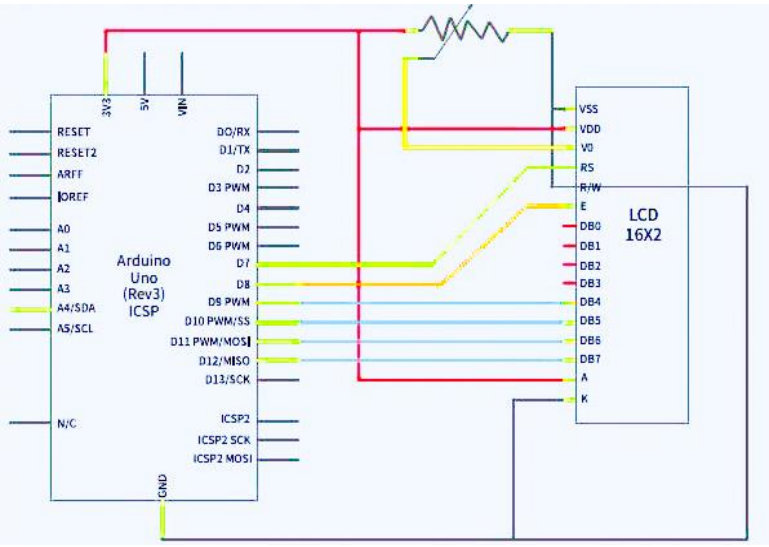


Fig. 3.3.1: Display connection schematic

To adjust the brightness of the display and control its contrast, it is also necessary to use a 10 kΩ potentiometer, connected through the V0 pin. Care must be taken to ensure that the yellow cable between the potentiometer cursor and pin 3 (V0) of the display is well connected, otherwise the potentiometer would not work and the display would appear to be off. [17]

3.3.2 RGB Led

The acronym LED stays for light emitting diodes. Traditional LEDs are those that can emit only one type of light, usually white (Fig. 3.3.2a).

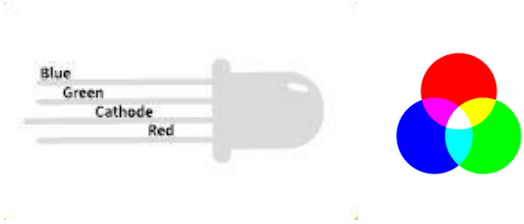


Fig. 3.3.2a: RGB Led

At first glance, the RGB (red, green, blue) LED looks like a normal LED. However, inside the usual LED, there are actually three LEDs, one red, one green and one blue. This could be very useful for visual signaling in testing phase, since, based on the color of the LED, it is possible to discriminate between ranges of distances.
By controlling the brightness of each of the individual LEDs you can mix the different intensities to get any color you want. By overlapping different colors and intensities, we can obtain a light characterized by different shades.

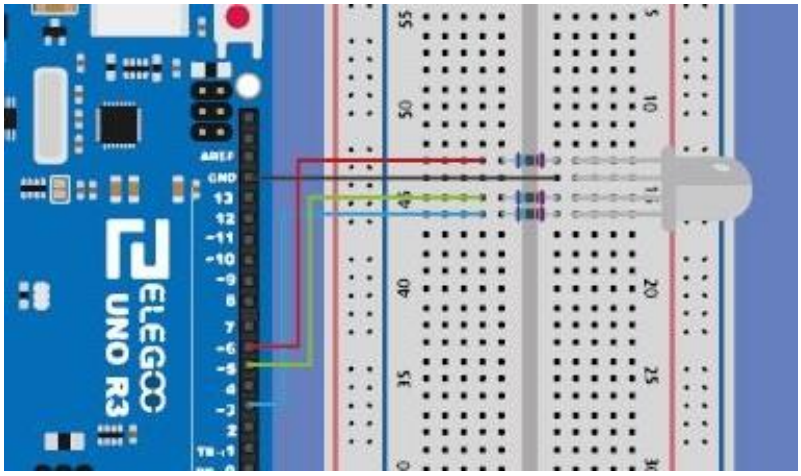


Fig. 3.3.2b: LED connection schematic

There are 4 wires, three of which are the positive connections of the LEDs while the fourth and last wire (common cathode) represents the negative connection and is common to all three LEDs. The common cathode is easy to recognize because it is the longest cable. Each internal LED requires a 220 Ω resistor to prevent the flowing of too much current into the LED, therefore, the three positive LED cables (corresponding to the three colors) are connected to the digital output pin of the UNO R3 board through these resistors. [18]

3.3.3 Passive buzzer

The buzzer is an audio signaling device. For a real-time system that aims to improve the safety of two-wheeled vehicles, the acoustic signaling is much more important than the visual one and was therefore fundamental both in the debug phase and in the sensor test phase.

Passive buzzers, unlike active buzzers, do not have an internal oscillator: they need an external frequency power source. In particular, to use a passive buzzer, it is necessary to provide it with a square wave which in Arduino we obtain with a PWM signal. The ability to control the frequency of the PWM signal allows us to reproduce notes or sounds by vibrating the air. Different vibrations generate different sounds. The buzzer has only two cables: positive and negative. [19]

To connect the buzzer to the UNO R3 board, the positive (red) part is connected to a digital pin, while the negative (black) part is connected to GND, as shown in Figure 3.3.3.

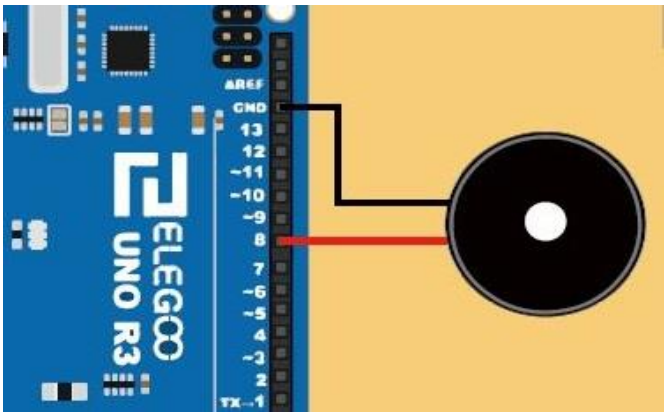


Fig. 3.3.3: Buzzer connection schematic

3.3.4 SD Card Reader

The SD card reader (Fig. 3.3.4) allows to facilitate applications that require the use of an external memory to save data. In an application like this one, it is essential if we want to have a data warehouse to analyze. Through the display it is possible to view the measurements in real-time but, to analyze the data, detect any errors and to perform the tuning of all the sensor parameters it is more efficient to register a database.

This module allows a simple interface with the Arduino board and can be used as a simple peripheral. Through appropriate programming of the card, it is possible to read and write to an SD memory. The device uses the SPI protocol to communicate with the microcontroller.

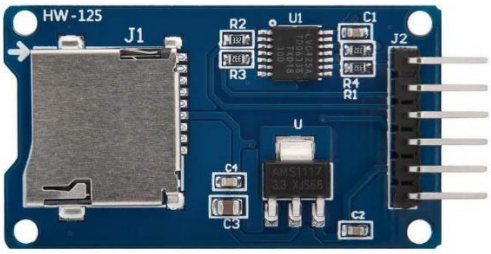


Fig. 3.3.4: SD card reader module

The SD card reader has, thus, 6 pins:

- VCC pin – power supply pin to be connected to the 5 V of the UNO R3 board.
- GND pin – to be connected to the microcontroller GND pin.
- MISO pin – device data output.
- MOSI pin – device data input.
- SCK pin – Clock pin generated by the Elegoo UNO board for data synchronization.
- SS pin – used by the board to enable the devices if there are more peripherals in communication.

The first two pins are used to power the peripheral while the other 4 are used for communication with the microcontroller. The SPI (Serial Peripheral Interface) interface is a type of serial communication that occurs between a "master" device (the UNO R3 board) and one or more devices called "slaves" (SD card reader). Data transmission on the SPI bus is based on the shift registers operation. Each device, both master and slave, is equipped with an internal shift register whose bits are output and, simultaneously, entered, respectively, through the MOSI output (Master Out Slave In) and the MISO input (Master In Slave Out) and they are synchronized via an SCK clock signal. [20]

3.3.5 Power Bank

To correctly simulate the operation of the distance sensor it was necessary to test it on a bicycle. The UNO R3 board has an external power jack available through which it is possible to power the microcontroller with a 9 V battery but for greater convenience we used a power bank connected through the USB port.
The power bank has a capacity of 20,000 mAh ensuring a power of 45 W. It has several input / output ports:

- **Input** Micro-USB: DC 5 V / 2 A
- **Input** USB di type-C: CC 5V/3A, 9V/2A, 15V/2A, 20V/1,5A
- **Output** USB di type-C: CC 5V/3A, 9V/3A, 12V/3A, 15V/3A, 20V/2.25A
- **Output** USB 1-2: DC 5V/3.4A (in total). Max 2.4 A for each.

The power bank used (Fig. 3.3.5) also has the "power delivery" (PD) function which guarantees the possibility of supplying any appropriate voltage/current combination among those available (up to a maximum power of 45 W), simply by communicating with the powered device. [21]

Fig.3.3.5 Power Bank

3.4 Final hardware architecture

Connecting all the electronic components together we got the complete final circuit in Figure 3.4b. Almost all digital pins of the microcontroller were used. The breadboard was only half powered (on 2 columns only), while the other half was left without power so that the microcontroller could also be placed there to make the prototype more compact.
As you can see in Figure 3.4b, the sensor cables have been reinforced with tape as being thinner wires than normal male-to-male cables, they can come off very easily. Reinforcing the connection thus avoids receiving incorrect values from the sensor due to the disconnection of some wires. The yellow and orange cables were not connected, as mentioned in the previous paragraphs, because they were not necessary for the I^2C serial connection. For a faster recognition of the connections, the colors red (5 V) and black (GND) have been used for all the power connections while, for the rest, different colors have been used according to the electronic components in order to recognize them more easily in the breadboard.
Obviously, the prototype size, a little rough, is due to the presence of the breadboard and power bank. If we consider that in an electric vehicle the power is supplied by the integrated battery and the circuit is printed on a PCB of a few centimeters, the dimensions are considerably reduced, thus obtaining a small, compact and light device.
The block diagram of the final architecture is shown below.

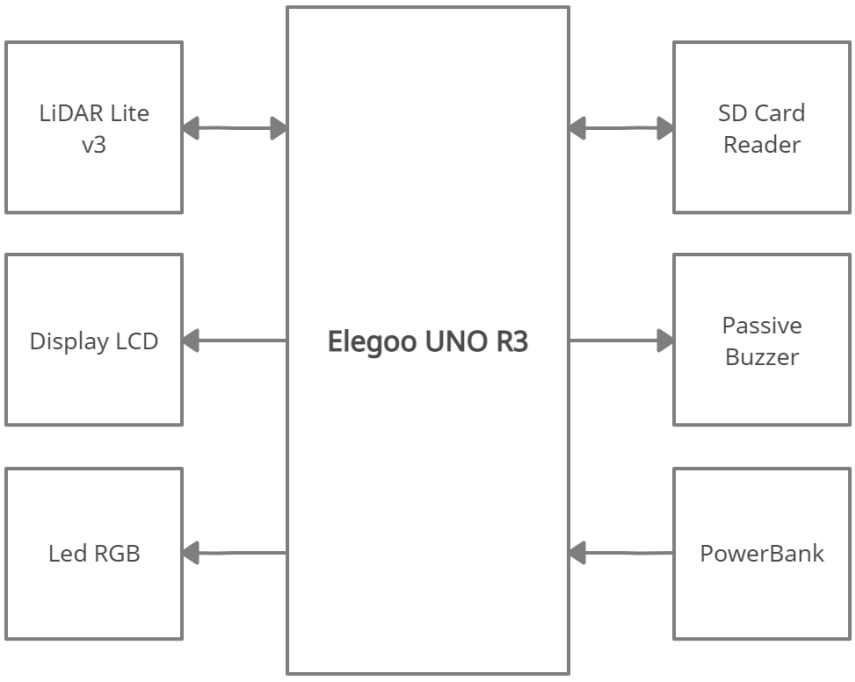


Fig. 3.4a: Schematic Block Diagram

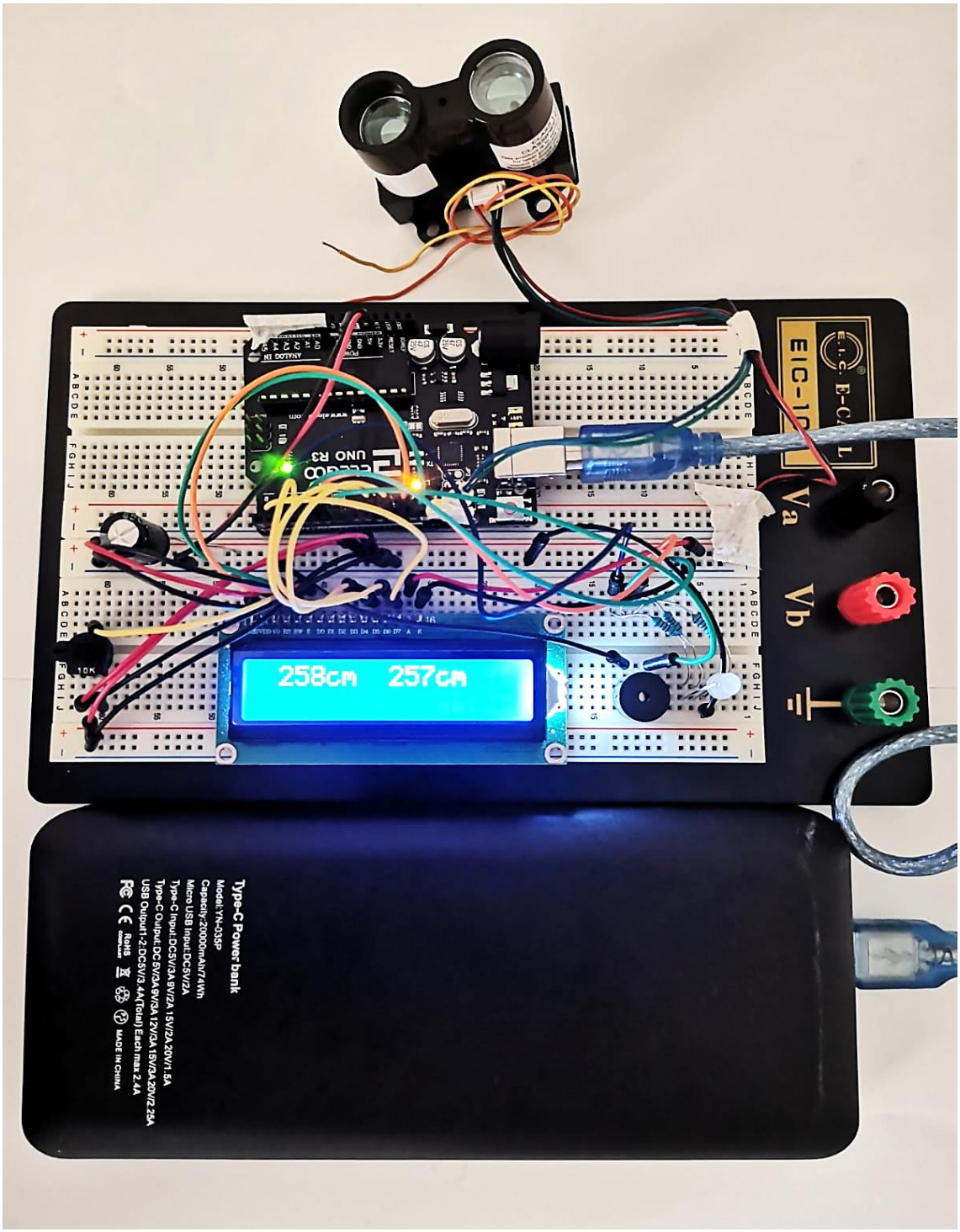


Fig. 3.4b: Prototype photo

Chapter 4

Software Programming

In the previous chapters we studied and analyzed the main characteristics on which the project is based and we created a hardware prototype that connects all the electronic components with each other.
In this phase of the project, however, the configuration of all devices is described, the way to be able to communicate correctly with the microcontroller, and the function that each of them must perform inside the sensor is established.
Programming is closely linked to the logic of what is going to be created and how it works. This, in fact, is an implementation phase in which we created the software (the mind of the sensor), which will then be loaded into the microcontroller to make the sensor work.
Before delving into the arguments related to the study of the sensor operating logic, we will start with the initialization and configuration of all the devices that make up the system.

4.1 Microcontroller configuration

Elegoo Uno R3 is a hardware platform consisting of a series of electronic boards equipped with a microcontroller. It is combined with a simple integrated development environment, Arduino IDE, for programming the microcontroller. All software is open-source and freely available on the web, and circuit diagrams are freely distributed.
In some cases, the board's microcontroller is pre-programmed with a bootloader which makes it easy to load programs onto the board's built-in flash memory. The ATmega328, for example, comes pre-programmed with a bootloader that allows you to load new code onto it without the use of an external hardware programmer. It communicates using the original STK500 protocol. It is also possible to bypass the bootloader and program the microcontroller via the In-Circuit Serial Programming (ISP) header using Arduino ISP or similar.
Conceptually, programs are loaded on all boards through a serial port. Elegoo UNO, on the other hand, is managed via USB, using an Atmega832 microcontroller programmed as a USB-serial converter. The Rx and Tx LEDs on the board flash when data is transmitted via the USB to serial chip and via the USB to computer connection.
The data remains stored on the board and can only be deleted by rewriting new data or by performing a reset operation. [22]

4.1.1 Development environment

The Arduino IDE (Integrated Development Environment) is a cross-platform application written in Java. To allow the drafting of the source code, the IDE includes a text editor able to compile and load the working and executable program on the board with a single click.
This development environment is provided with a C/C ++ software library, called "Wiring": the availability of the library makes it much easier to implement common input/output operations via software.
Arduino programs are written in a language derived from C/C ++. When opening the Arduino programming software and creating a new sketch, the programmer is only required to define two main functions, in order to create an executable file (Fig. 4.1.1):

- `void setup ()` – *function invoked only once, at the beginning of a program, and used for the initial settings that will remain unchanged during the execution of the program;*

- `void loop ()` – *function invoked repeatedly, whose execution is interrupted only when the power supply to the board is cut off.*

The term "void" indicates procedures which must not return anything and which are used to introduce functions. Between the braces you have to write the code with the commands to be executed on the microcontroller.
In the setup function, all the necessary information is given to the board before running the program. For example, it is possible to set some ports of the board as input or output, or to initialize the various devices. It may also contain commands to be executed by the microcontroller, but they are executed only once (at the beginning of the program).

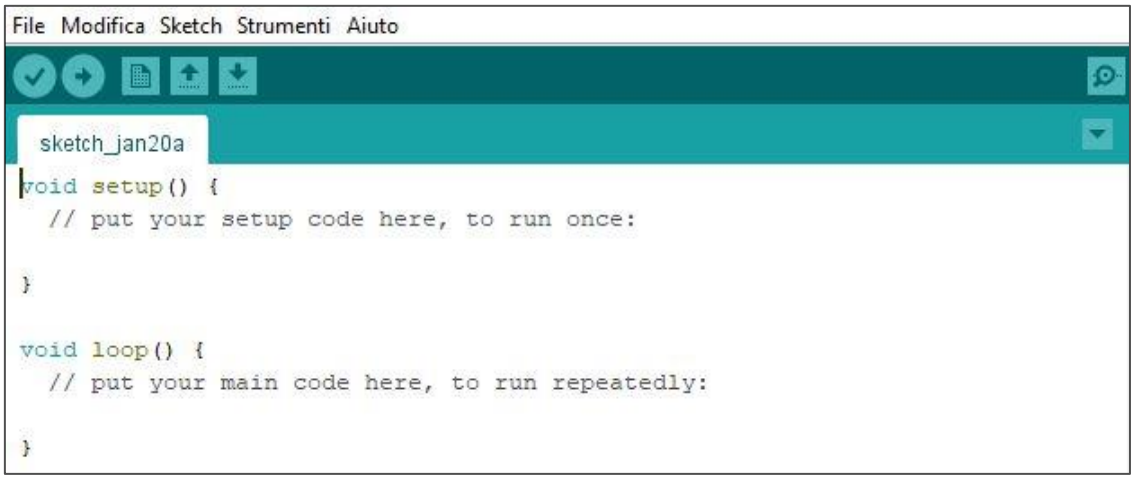


Fig. 4.1.1: Arduino IDE

In the loop function, on the other hand, there are all the information relating to the execution of the program and all the commands related to it. This function represents the heart of the program, the commands inside it are always repeated in order and when the code ends, it starts all over again (as in a loop). Once loaded on the UNO R3 board, therefore, the program will continue to run as long as the device is connected to the power supply. [23]
To see the results of a program running on the microcontroller, from the IDE, it is possible to activate a serial monitoring window on which we can see the output of instructions embedded in the program itself. To do so, we use the function:

```
Serial.print(parameter);
```

The serial monitoring function is very important and will be used a lot in the configuration phase of the Lidar sensor and, above all, in the debugging phase of all the electronic components.

4.2 Sensor programming

Some libraries are already incorporated in Arduino but, for this project, to simplify the programming of the LIDAR-Lite v3 sensor and for its correct functioning, we added another one (*"LIDARLite.h"*) provided by the manufacturer. The library was added with an import tool included in the IDE.
Before implementing the two functions described in the previous paragraph, the libraries must be included at the beginning of the code. The purpose of the software libraries is to provide a collection of functions ready for use, that is, reuse of code, avoiding the programmer having to rewrite the same functions or data structures every time and thus facilitating development and maintenance operations.

```
#include <Wire.h>
#include <LIDARLite.h>
```

Through the *"include"* instruction, the program recognizes the functions or data structures of the libraries that will be used by the programmer. After including the libraries, the global variables and data structures used in the project are declared and initialized.

```
//Globals
LIDARLite lidarLite;
int cal_cnt = 0;
```

The two slashes "//" are used to introduce descriptions or comments which are often useful for the programmer to recognize the code. These statements are invisible in the program and are not recognized as code by the compiler.
The first line of code creates the *"lidarLite"* object belonging to the *LIDARLite* class. The created object acquires all the properties of this class. From the programming point of view, it is possible to say that a class acts as a type for a certain object belonging to it. This type of programming is called object-oriented programming.
A variable *"cal_cnt"* (calibration counter) of integer type was then created and it was initialized to the value '0'. This variable has the function of a counter and, later, during the description of the "*loop ()*" function, I will explain its operation.
At this point the *"setup ()"* function is configured by carrying out the initial settings. The serial connection between the computer and Arduino is initialized with the "*Serial.begin()*" instruction. Both sides of the serial connection, i.e., both the computer and the Arduino, need to have the same connection in order for the data to be read correctly. In this case the set transmission speed is "9600 bit/s" which is, usually, the default value for Arduino.

```
void setup () {

  Serial.begin(9600);

  lidarLite.begin(0, true);
  lidarLite.configure(0);
}
```

The other two instructions in the setup function concern the configurations of the Lidar sensor. The two functions *"begin ()"* and *"configure ()"* belong to the *"LIDARLite"* class of the homonymous library and are called through the lidarLite object. The *"begin ()"* instruction, for example, initializes the sensor by configuring it with 2 parameters:

1. The first parameter sets the device in configuration '0', or default configuration, which guarantees balanced performance between measurement speed and accuracy.

2. The second parameter determines the speed of the I^2C connection. By default, the base frequency of the I^2C connection is set to 100 kHz, but if "*true*" parameter is set, the frequency increases to 400 kHz allowing the sensor to work at a faster speed.

The second instruction, on the other hand, allows to configure the sensor using a series of pre-set configurations to choose from. This allows the programmer to optimize the distance sensor based on the type of project and on the use to be made of it. For now, the default configuration will be kept, but, in the next paragraph (4.2.2) all the ways in which it is possible to configure the device will be analyzed in detail: starting from the preset configurations up to the setting of individual registers that characterize the sensor properties.
After making the initial settings, the program enters the *"loop ()"* function, which contains the body of the whole program that will be executed continuously.
The *"loop ()"* function is the main Arduino function that contains all the program logic and all the functions used within it. The principal functions are listed below:

- *Distance ()* – function used for distance detection (including bias receiver correction)

- *Store ()* – function that takes care of saving the correctly performed measurements and filtering them.

- *Obstacle ()* – function that takes care of all the logic for obstacle detection

Other functions (less important) have been created to program the components used for debugging and testing phase. We will explain these functions in the 4.4 successive paragraph.

To get a clearer idea of the sequence of operations performed by the sensor for the entire obstacle recognition process, the Flow Chart of the main function *"loop ()"* has been shown below (Fig. 4.2a).

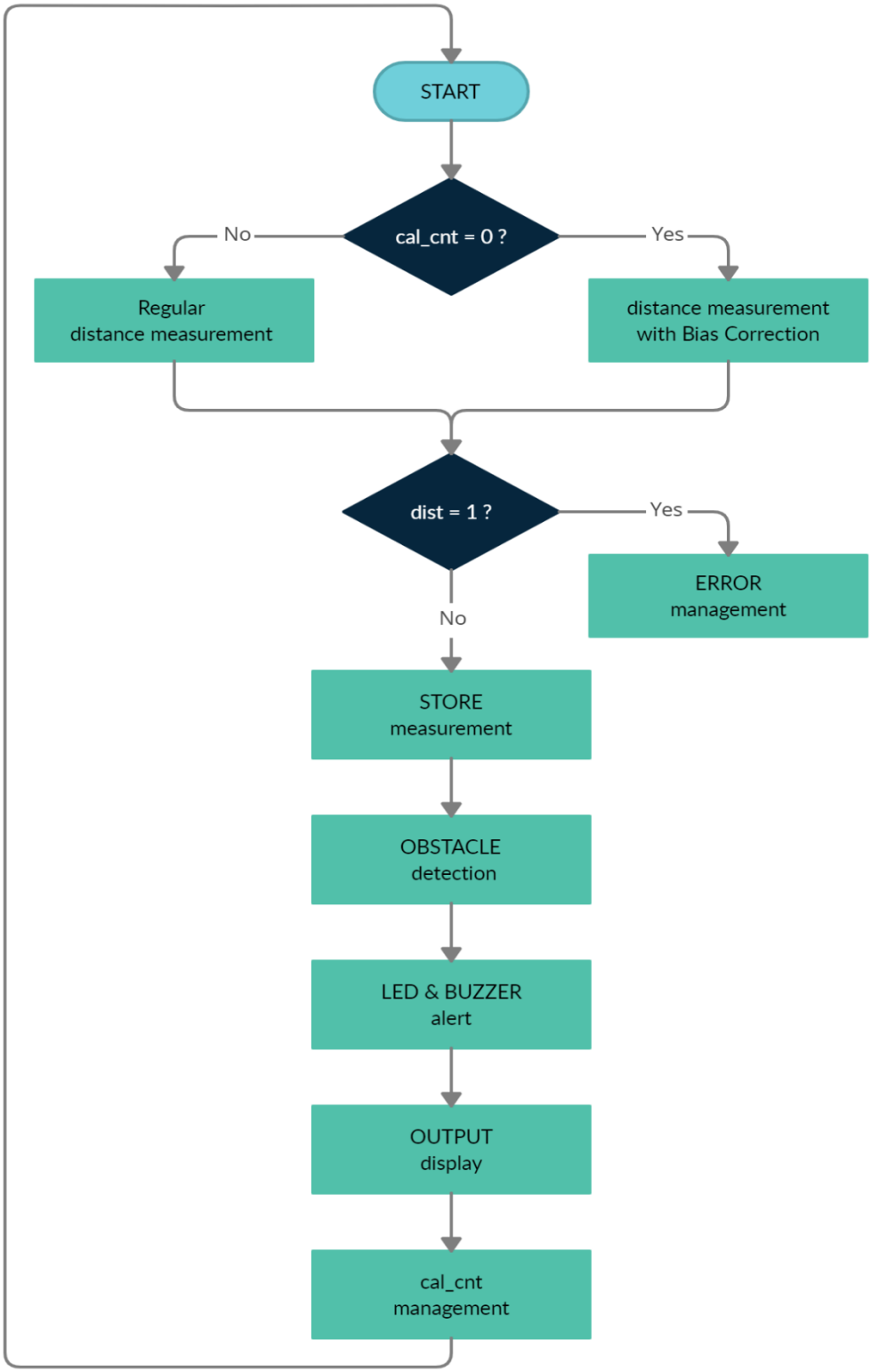


Fig. 4.2a: Loop function Flow Chart

At the beginning of the cycle, the integer variable *"dist"* (distance) is declared. It will be used to temporarily store, in each cycle, the measurement performed by the LIDAR-Lite v3 sensor.

```
void loop () {
  int dist;

  if (cal_cnt == 0) {
    // bias correction
    dist = lidarLite.distance();
  } else {
    // no bias correction
    dist = lidarLite.distance(false);
  }
}
```

The main part of the code begins with the conditional statement *"if ()"*.
The *"distance ()"* function, contained in the LIDARLite library, allows you to acquire the measurements performed by the sensor.
When this function is called, a distance measurement is performed and the result is stored in the variable *"dist".* The measurement is carried out through the following steps:

1. The value 0x03 (normal measurement) or 0x04 (measurement with correction at the receiver) is written in register 0x00 to start an acquisition.

2. Register 0x01 is read (through the *"read ()"* command) and if the first bit is '0' then the sensor is ready for measurement otherwise the sensor is busy and goes in loop until the first bit becomes '0'.

3. Two consecutive bytes are read and saved from register 0x8f.

4. The first value (high byte) is inserted into an array and shifted by 1 byte, then the second value (low byte) is added. The result of this operation is the distance measured in cm.

As already described above (paragraph 2.3.1), the Lidar Lite sensor can perform measurements by performing an accuracy correction at the receiver (receiver bias correction).
The "*if/else*" statement allows to choose between these two types of measurements based on the condition of the *"cal_cnt"* variable. Only and exclusively if the variable *"cal_cnt"* is equal to '0', the Lidar sensor performs a measurement with correction; in all other cases the measurement is performed without bias correction. To do so, you must enter the boolean parameter *"false"* when calling the function.
Since the variable *"cal_cnt"* is initialized to '0', it follows that the first measurement will be made with bias correction.

But how does the value of the *"cal_cnt"* variable change?

```
// counter increment
cal_cnt++;
cal_cnt = cal_cnt % 100;
```

After performing a measurement acquisition, the value of the *"cal_cnt"* variable (which has the counter function) is increased, thus passing from '0' to '1'.
The next line of code, on the other hand, is used to reset the counter when the variable reaches the value 100. In this case the (default) value '100' is set as the counter limit, but any other value can be set. Using the "%" operator, the *"cal_cnt"* variable is divided by 100 and updated with the division "remainder".
Through this logic, it follows that the value of the variable will be reset to the value '0' only and exclusively if it reaches the value '100', since from the division we obtain "quotient = 1" and "remainder = 0".
According to the logic of the program, therefore, a distance measurement with bias correction is initially performed, while the other 99 will be faster as they are performed without bias correction; from here, the same cycle will be repeated over and over. After acquiring the measurement and increasing the counter value, we used the *"Serial.print(dist)"* monitoring instruction which allows the programmer to display the value of the variable passed as a parameter on the screen.

```
// Output distance on display
Serial.print (dist);
Serial.println ("cm");
```

The second instruction, on the other hand, is used to complete the measurement with the measurement unit. The LIDAR Lite v3 sensor resolution (see Tab. 3.1d) is 1cm and, exactly for this reason, its measurements are reported in centimeters.

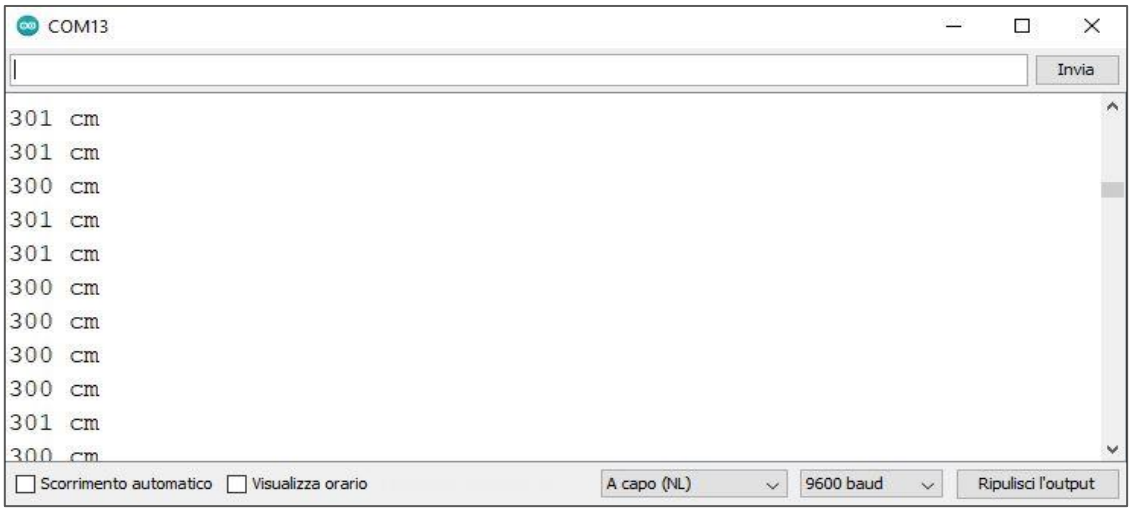


Fig. 4.2b: distance measurements on Serial Monitor

Before compiling the program, the serial COM port connecting the Arduino and the computer was set.
The program is then compiled and loaded on the UNO R3 board, where it will be executed in loop on the microcontroller. By opening the serial plotter function in the Tools section, it is possible to view all the measurements performed by the sensor (Fig. 4.2b).
In this case, test measurements were initially carried out, placing the sensor on the floor, with the beam directed perpendicular to the ceiling.

4.2.1 Control registers

The LIDAR Lite sensor can be configured with different parameters in order to customize the algorithm for measuring distances. As already mentioned in the previous chapters, this is a strong point of this device. Performance may vary based on the setting of control registers that allow the user (programmer) to configure the device by choosing the right compromise between speed, distance range and sensitivity. Below, we will analyze the use of some of the most important configurable registers. [15]

Note: Register values are encoded in the hexadecimal (and not decimal) system. This allows for a quick switch to binary code by substituting 4-digit groups. Each hexadecimal digit represents 4 bits, therefore, with 2 hexadecimal digits it is possible to change the value of an 8-bit register. All registers contain 1 byte and can be both read and written.

RECEIVER BIAS CORRECTION

Address	Name	Description	Initial Value
0x00	ACQ_COMMAND	Device command	--

The first register, ACQ_COMMAND, can be set to choose whether or not to have the receiver bias correction at the receiver. By omitting the correction routine, the device can perform distance measurements faster.
Unfortunately, accuracy and sensitivity are adversely affected if environmental conditions change (e.g., target distance, device temperature and optical noise). To have good performance, it was preferred to perform the receiver bias correction of the receiver periodically by doing it at the beginning of 100 sequential measurements (as recommended).
This register also has a very important reset function; in fact, by setting the value 0x00 in the register, it is possible to reset all the registers to the initial default value. [15]

MAXIMUM ACQUISITION COUNT

Address	Name	Description	Initial Value
0x02	SIG_COUNT_VAL	Maximum acquisition count	0x80

The maximum acquisition count limits the number of times the device will integrate acquisitions to find a correlation record peak (from a returned signal), which occurs at long range or with low target reflectivity. The LIDAR-Lite v3 sensor, in fact, performs a measurement after making multiple acquisitions. It is logical to think that a greater number of acquisitions allows to reduce the percentage of measurement error, especially when using at high range or if the target hit by the laser beam does not correctly reflect the signal.
In our case, the sensor pointing distance will be kept constantly at 10 m, but we don't know the reflectivity of the target which could vary according to the material it is made of.
Therefore, it could be useful to increase the value of this register to increase the probability of receiving correct measurements, and above all, more accurate. Increasing the value of this parameter, however, increases the response time, therefore, it is necessary to find the right compromise in setting all the parameters. [15]

MEASUREMENT QUICK TERMINATION DETECTION

Address	Name	Description	Initial Value
0x04	ACQ_CONFIG_REG	Acquisition mode control	0x08

The ACQ_CONFIG register is an enabling register that allows you to activate or deactivate other registers with different functions. The logical value of each bit involves the enabling of multiple registers and we paid attention to two of these bits:

Bit	Function
3	Measurement quick termination
5	Measure delay

You can enable quick-termination detection by clearing "bit 3" (setting it to '0' value). The device will terminate a distance measurement early if it anticipates that the signal peak in the correlation record will reach maximum value. This allows for

faster and slightly less accurate operation at strong signal strengths without sacrificing long range performance.
Bit 5, on the other hand, allows you to choose the delay to be used for measurements. The default delay (0xc8) between two automatic measurements corresponds to a repetition frequency of 10 Hz. By configuring this register to '1', you enable the use of the *MEASURE_DELAY* register where you can set a specific value: for example, 0x14 corresponds to a repetition of 100 Hz. A low delay results in a high repetition frequency of the measurements. [15]

Address	Name	Description	Initial Value
0x45	MEASURE_DELAY	Delay between automatic measurements	0x14

It must be taken into account that the delay is calculated after the measurement has been completed. This means that the duration of the measurement (which can vary with the signal strength) affects the repetition rate. Therefore, if greater speed is desired, we will not only need a low delay but we will also need to limit the maximum number of acquisitions.

DETECTION SENSITIVITY

Address	Name	Description	Initial Value
0x1c	THRESHOLD_BYPASS	Peak detection threshold bypass	0x00

The default valid measurement detection algorithm is based on the peak value, signal strength, and noise in the correlation record. This can be overridden to become a simple threshold criterion by setting a non-zero value.
To have a high sensitivity with frequent incorrect measurements, we can leave the default value '0' or we can set it to a not too high value, such as "0x20". On the other hand, to have a reduced sensitivity and have fewer incorrect measurements, it is better to set a higher threshold value, such as "0x60". Care must be taken to set the threshold. If it is too high there is the risk that the sensor will not be able to carry out all the measurements correctly. [15]

4.2.2 Configuration of the registers

The LIDAR-Lite v3 sensor, in addition to being widely configurable by setting its control registers, also has several configurations preset, included in the library, each with ad hoc instructions according to the application in which the user needs.

There are 6 configurations preset and they are identified by numbers:

0. *Default mode* – It is the default configuration and it allows balanced performances.

1. *Short range, high speed* – It is optimized for short range measurements and increases the sensor speed.

2. *Default range, higher speed short range* – It is used for all measurements but is optimized for short range ones.

3. *Maximum range* – Uses "0xff" maximum acquisition count to optimize the sensor for long range measurements.

4. *High Sensitivity detection* – It overrides default valid measurement detection algorithm, and uses a threshold value for high sensitivity and noise.

5. *Low Sensitivity detection* – It overrides default valid measurement detection algorithm, and uses a threshold value for low sensitivity and noise.

Each of these configurations optimizes the sensor in different ways according to the user's needs.
In our project there is a need for a high detection speed, since this allows for a greater number of measurements. The quantity of measurements affects the correct detection of obstacles and, above all, the vehicle speed must be taken into consideration: the sensor, in fact, is not stationary.
However, as described in chapter 1, the sensor positioning limits strongly affect the quality of the measurements. An inclination angle of only 5/6 ° greatly influences both the quantity of rays reflected back from the ground, due to the diffusion properties of the light, and the accuracy of the measurements performed. In fact, with a difference of only a few degrees between the laser beam direction and the ground plane, small differences in the sensor installation accuracy would be enough to compromise the accuracy of the measurements.
Furthermore, the different reflectivity of the obstacles affects the return signal strength and this is a not negligible problem that forces us to privilege the accuracy of the measurements.
To have a more detailed description of the environment and the obstacles that may be encountered along the route, therefore, it is necessary to make as many measurements as possible but, above all, to make as many acquisitions as possible to improve the accuracy of the measurements performed.
A compromise had to be found in sensor configurations and control register settings. Initially, in order to increase the accuracy of the measurements, the "Maximum range" mode was chosen among the configuration presets. The latter optimizes the LIDAR-LITE v3 sensor by maximizing the number of acquisitions made for a measurement.

```
// Maximum range
write (0x02, 0xff, lidarliteAddress);
write (0x04, 0x08, lidarliteAddress);
write (0x1c, 0x00, lidarliteAddress);
```

In particular, the value of the *SIG_COUNT_VAL* register (address 0x02), is set to the value "0xFF" (which corresponds to 255). The "maximum acquisition count", therefore, has been set to the maximum allowed number. It should be noted that, to take a measurement, 255 acquisitions can be made but this does not mean that this happens for all measurements. In fact, the value '0' has been set in bit 3 of the *ACQ_CONFIG* register (address 0x04) which allows the measurement to be terminated earlier if the sensor detects a high signal correlation.
In this way, if the sensor detects a peak in signal correlation, it quickly completes the measurement, otherwise, it performs as many acquisitions as possible to improve the accuracy of the measurement.
With these configurations, on the one hand the accuracy of the measurements has been improved but, on the other hand, the speed of the sensor is conditioned by the number of acquisitions performed to complete a measurement.
Unfortunately, with the following configuration I found a considerable reduction of the measurement debugging the sensor, so I chose to set the "default" mode and to modify the "maximum acquisition count" register with the value "0x90" that is greater than the default one (0x80) but not excessively high.
This modification slows down the sensor speed, but not too much, so it was possible to compensate for this constraint by activating the "*MEASURE_DELAY*" mode. The latter was set by enabling the register of the same name (address 0x45) and setting bit 5 in the *ACQ_CONFIG* register to '1'.
Increasing the repetition rate of measurements speeds up the sensor without affecting the accuracy of the measurements.
The necessary settings for our device can therefore be made by adding only two instructions in the *"setup ()"* function of the program.

```
write (0x02, 0x90, lidarliteAddress);
write (0x04, 0x28, lidarliteAddress);
```

"0x90" represents the maximum acquisition number for each measurement while "0x28" is the value that the *ACQ_CONFIG* register assumes when you need to enable quick termination mode and set the use of the *MEASURE_DELAY* register.

4.3 Measurement errors

In chapter 1, I computed the pointing distance of the sensor; this distance, in ideal conditions (perfectly flat road), should always be constant and therefore maintain the value of 10 m.
Unfortunately, the behavior of the sensor in reality differs significantly from that in ideal conditions. In fact, the operation of the sensor is subject to many factors that can affect the accuracy in measuring distances.

4.3.1 Data filtering

First of all, two-wheeled vehicles are very unstable and this characteristic creates, in turn, instability in the sensor as well. While the vehicle is in motion, it is subject to oscillations along the axis normal to the ground caused by road irregularities, but also along the parallel axis due to small oscillations that the driver carries out while driving. The latter is a well-known instability problem while moving in light two-wheeled vehicles.
This leads to measurement errors that cause the pointing distance value to fluctuate, while it should be constant and equal to 10 m. Furthermore, as explained in Chapter 1, the angle of inclination of the sensor is 5/6 °; this means that the laser beam is almost parallel to the ground plane and small oscillations would be enough to generate large variations in the measurements.
Figure 4.3.1a shows the trend of the sensor measurements traced by the Arduino plotter which highlights the variation from the ideal and constant value of 10 m.

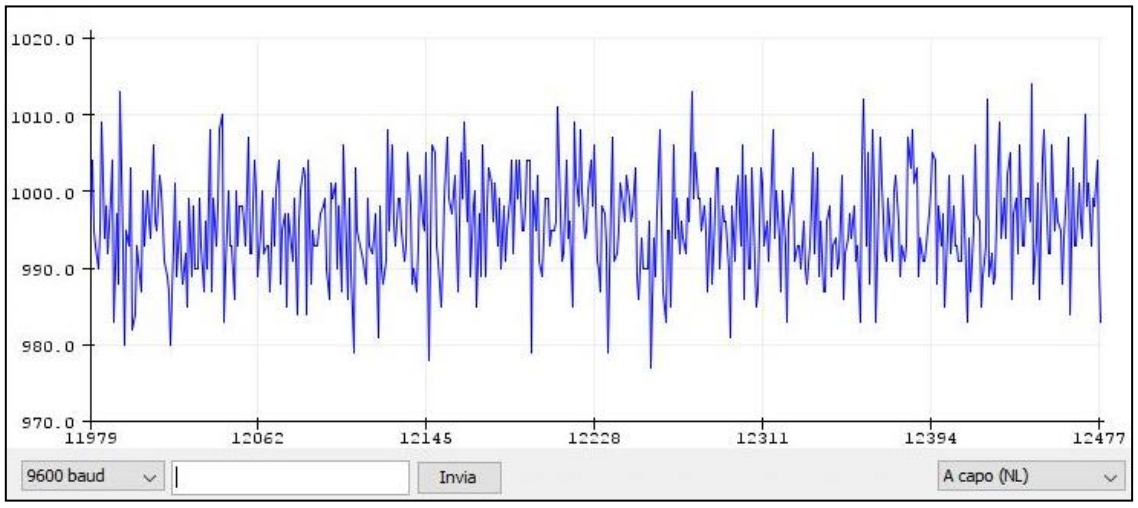


Fig. 4.3.1a: Trend of measurements traced by serial plotter

As shown in the graph, there are many peak values which could be a problem for the correct functioning of the obstacle detection logic. In addition to this, there is an offset of 3-4 centimeters due to an error in mounting and fixing the sensor on the

vehicle (this because is a prototype). The accuracy of the measurements is ± 15 cm and is a high uncertainty considering that the vehicle is still stationary. To solve this problem, a data filtering algorithm was used in order to reduce the error percentage of the measurements and to have more stable values closer to the theoretical 10 meters.

"What does the data filtering algorithm consist of?"

The mathematical procedure of the "moving average" was used to filter the data. Through a conditional instruction in the *"loop ()"* function, the *"store ()"* function is called. It allows us to temporarily record the measurements.
Initially, we used an algorithm based on a *"for ()"* loop, but later the code was optimized with a better algorithm which allowed to obtain about 5 more measurements per second.
Whenever a measurement is received, it is stored sequentially in an array, exploiting the *"store_cnt"* variable used as a counter to scroll the position in that array.
Measurement after measurement, within the *"store ()"* function, an algorithm is used to generate the average value through the values contained in the array.

```
distAvg = (distAvg * SAMPLE_NUM) - distArray[store_cnt];

distArray[store_cnt] = distance; // Store measurements

distAvg = distAvg + distArray[store_cnt];
```

Each time the function is called, the oldest measurement in the array is subtracted from the sum *"distAvg"* of all measurements, then instead, the newest one is saved in the array and added to the other measurements. The result is saved updating the variable itself. At the end, the value of the variable represents the sum of all the measurements in the array.
By dividing the obtained value by the number of *"SAMPLE_NUM"* measurements present in the vector, we obtained the average value of the measurements.

```
distAvg = distAvg / SAMPLE_NUM; // Average Computation
```

The time window considered, therefore, is not fixed. As soon as a new measurement is inserted into the array, it replaces the older one. In this way there is not a fixed average based on a certain set of values, but a moving average.
In the calculation of the moving average, the number of elements is fixed, but the time window considered advances: the oldest measurement in the series is gradually replaced with the new one. This mathematical procedure causes the average to move in progression with the trend of the measurements.

Thanks to the calculation of the moving average, it is possible to have a much cleaner "*distAvg"* average value by reducing the noise. Figure 4.3.1b shows the trend of the average value plotted by the plotter.
Considering the average of the measurements instead of the effective value, it was possible to obtain more compliant measurements which allow to limit the influence of incorrect acquisitions due to any possible oscillations of the vehicle.

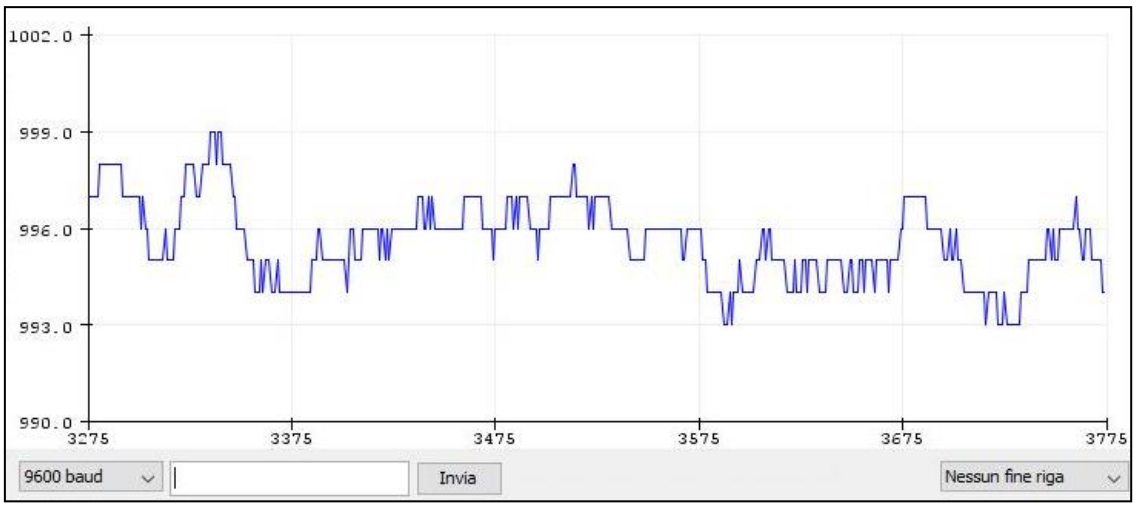


Fig. 4.3.1b: Trend of filtered measurements traced by serial plotter

If you compare the results obtained by filtering the measurements with those shown in Figure 4.3.1a (without filtering), you immediately notice a radical difference. First of all, there are no more peak values, but above all, the measurements accuracy has improved a lot. The uncertainty has reached a value of ± 3 cm (five times lower than before) which is more than acceptable.

4.3.2 Returned signal strength

In addition to instability errors, measurements can also be affected by errors due to incorrect signal reception. The correct reception of a reflected signal is strongly influenced by several factors:

- **TARGET DISTANCE**
  The relationship of distance (D) to returned signal strength is an inverse square. So, with increase in distance, returned signal strength decreases by $1/D^2$ or the square root of the distance.

- **TARGET SIZE**
  The relationship of a target's Cross Section (C) to returned signal strength is an inverse power of four. The device transmits a focused laser beam that spreads at a rate of approximately 0.5° as distance increases. The

approximate beam spread in degrees can be estimated by dividing the distance by 100. When the beam is larger than the target, the signal returned decreases by $1/C^4$ or the fourth root of the target's cross section. The diameter of the laser beam at a distance of 10 m is "10 cm". According to these considerations, therefore, it will be difficult to detect obstacles smaller than 10 cm because the returned signal strength is reduced by an amount that depends exponentially on the size of the obstacle.

- **ASPECT**
  The aspect of the target or its orientation towards the sensor affects the observable cross section and, therefore, the amount of returned signal decreases as the aspect of the target varies from the normal.

- **REFLECTIVITY**
  Reflectivity characteristics of the target's surface also affect the amount of returned signal. These characteristics can be divided into three categories:

  1. *Diffuse Reflective Surfaces*
     To this category belong all those materials which, instead of sending the laser ray back at a precise angle, cause a uniform dispersion of the incident ray. These materials can be read very well by the sensor.

  2. *Specular Surfaces*
     Specular surfaces can be found in those materials that have a smooth quality that reflect energy instead of dispersing it. It is usually very difficult or impossible for the device to recognize the distance of many specular surfaces. Reflections off of specular surfaces tend to reflect with little dispersion which causes the reflected beam to remain small and, if not reflected directly back to the receiver, to miss the receiver altogether. The device may fail to detect the object unless viewed from the normal.

  3. *Retro-reflective Surfaces*
     Retro-reflective surfaces reflect radiation in the same direction as the source with minimal dispersion and are very efficient even with wide ranges of incidence angles. An example of a retro-reflective surface is the reflector.

In summary, the correct detection of an obstacle depends on many factors and, most of the time, on the characteristics of the object itself and on how it is placed when the sensor beam hits its surface. A small and poorly reflective object, for example, can be very difficult to detect if it is distant. Due to the small inclination angle, in fact, the probability that the signal will be reflected back in the same direction is very low, with the risk of receiving a rather weak returned signal. [15]

If the sensor receiver does not detect the acquisitions correctly, there is the possibility that the sensor cannot perform a measurement, thus giving the “Error value” at the output (encoded with a "1 cm" measurement) which indicates a failed measurement.
To prevent incorrect measurement detection from affecting the data filtering function, I inserted an “*if ()”* statement to exclude incorrect measurements from the filter. Values of “1 cm”, could strongly influence the average value of distance measurements, especially if there are surfaces that are difficult to detect and for which many measurements fail. On the other hand, however, these measurements are very important to analyze the possible malfunction of the sensor in the various cases of use.

```
if (dist == 1) {
    Error = true;
    err_cnt ++;
} else {
    Error = false;
    err_cnt --;

    if (err_cnt < 0) {
      err_cnt = 0;
    }
}
```

The number of failed measurements is, therefore, stored in a variable *“err_cnt”* (used as a counter) in order to calculate the percentage of failed measurements. If a “1 cm” distance is recognized, the *“Error”* flag is enabled and the counter is incremented. If, otherwise, a different distance is detected (correct measurement) the counter is decremented. When the *“err_cnt”* value becomes negative, it is reset to ‘0’.
In addition, the counter is automatically reset to ‘0’ every 100 milliseconds to keep the effectiveness of the measurements updated in the time frame.
In the test phase, this function will be very useful to analyze the correct functioning of the sensor and to discriminate how the different factors affect the detection of distance measurements.

4.4 Other components programming

In the *"loop ()"* function there are other secondary functions used to control the operation of the other components used. Mainly, these components are used in the debugging and testing phase.

4.4.1 Display programming

To communicate with the LCD display, the *"LiquidCrystal.h"* library was included before the *"setup ()"* function and the "lcd" object belonging to the *"LiquidCrystal"* class was created. This way we initialize the library by associating any needed LCD interface pin with the Elegoo board pin number it is connected to.

```
#include <LiquidCrystal.h>          // Lcd library
LiquidCrystal lcd (RS, E, D4, D5, D6, D7);
```

Subsequently, the *"begin ()"* function (available in the library) was used to initialize the LCD display in the *"setup ()"* function. The two input parameters in the function are used to set the number of columns and rows of the LCD (16 columns, 2 rows).

```
lcd.begin (16, 2);            // LCD initialization
```

In the loop, the *"print ()"* function included in the LCD library was used to send data output to the LCD display, whenever needed.

```
lcd.print (dist);
lcd.print ("cm");
```

4.4.2 Led programming

The RGB led allows us to have a visual feedback both in the debug phase and in the test phase. To program the LED, in the *"setup ()"* function we need to define the three pins that have been used as output. The *"pinMode ()"* function is used to configure the specific pin as an input or output. Through the *"digitalWrite ()"* function it is possible to switch on or off the single internal LEDs (HIGH=ON, LOW=OFF)

```
pinMode (RED, OUTPUT);        // red LED initialization
digitalWrite (RED, HIGH);     // Set default configuration
```

In the *"loop ()"* function, we used the *"analogWrite ()"* function not only to turn on the LEDs but also to set the intensity of the LEDs. The input parameter can assume a value between 0 and 255 and by mixing the various intensities of the three LEDs, different colors can be obtained, as mentioned in paragraph 3.3.2.

```
analogWrite (RED, 127);     // Set default configuration
```

For the obstacle detection logic, the *"switchLeds ()"* function was used. Different colors were used according to the different types of obstacles detected. Therefore, a different color is assigned to each value of the *"flag"* variable. For example, if *"flag"* has the value '1', the green color is set, while, if *"flag"* has the value '2', the yellow color is set and so on.

```
void switchLeds () {        // Function to manage LEDs

  if (flag == 1) {...}    // green = road bump case

  else if (flag == 2) {...}  // yellow = step up
  ...
}
```

4.4.3 Buzzer programming

The buzzer is used for acoustic signaling. Here too, in the *"setup ()"* function we need to define the pin that was used as output and through the *"pinMode ()"* function we configured the specific pin as output.

```
pinMode (BUZZ, OUTPUT);       // buzzer initialization
```

In the *"loop ()"* function, on the other hand, the *"buzzer ()"* function was used to generate an acoustic frequency with a 50% duty cycle. To set the intensity of the signal, we used the *"alert ()"* function which receives the variable *"x"* as input. The latter can be set with different values based on the obstacle type. For example, the lower the value of *"x"* the more the frequency and, therefore, the alert intensity will increase.
In our logic, the value of *"x"* will depend directly on the value of the obstacle distance the way that if is very distant, there will be a high value of *"x"* and, therefore, a low intensity alert will occur, and vice versa.

```
void alert (int x) {        // x = frequency management

    if ((cal_cnt % x) == 0)
        buzzer ();
}
```

4.4.4 SD card reader programming

Due to the absence of available digital pins, in the test phase, the SD card reader was connected to the microcontroller by disconnecting the LED and the buzzer. In fact, the SD card reader uses the SPI serial communication, available only on digital pins 10, 11, 12 and 13. For the correct functioning of the device, we added two libraries: the *"SPI.h"* library for communication and the *"SD.h"* library for the device management. First of all, therefore, a *"fileToWrite"* object belonging to the *"File"* class has been created.

```
#include <SPI.h>            // SPI library
#include <SD.h>             // SD library
File fileToWrite;
```

In the *"setup ()"*, we employed the *"begin (10)"* function (using the SD library) to initialize the device. The input parameter "10" represents the digital pin to which the "CS" signal of the reader is connected.

```
SD.begin (10);      // SD card reader initialization
```

All the code used for file writing on the SD card has been enclosed in the *"SDwriter()"* function. The latter allows you to open the chosen file using the *"open ()"* function, verify its correct opening and, if so, write all the information we need, exactly as if you wanted to send data to the serial monitor. At the end the file is closed simply using the *"close ()"* function.

```
void SDwriter () {

  //file opening in writing mode
  fileToWrite = SD.open ("data.txt", FILE_WRITE);

  if(fileToWrite) {     //If the file was correctly opened

    fileToWrite.print(distAvg);    //file writing
    ...
    fileToWrite.close();         //file closing
  }
}
```

4.5 Obstacle detection logic

After configuring the sensor in the best possible way and writing the algorithm that allows us to refine the measurements made by the sensor, I paid attention to the obstacle detection logic.
The sensor we are going to develop should detect obstacles in real time and instantly alert the driver so that he has time to notice the obstacle and avoid it or stop before hitting it. To correctly notify obstacles it is necessary to know the speed of the vehicle and to have a synchronous system capable of providing us with constant measurements over time.
To correctly perform the calculations useful for detecting obstacles, it is very important to understand how many measurements the sensor performs in a given time ΔT, in order to understand how accurately it is possible to detect obstacles.
We need ΔT to be constant and, therefore, all the measurements made by the sensor to be synchronous. After writing all the programming code, it may happen that the sensor takes different times between one measurement and another. The causes of this can be different:

- The detection of some obstacles may be more difficult due to the material type of the obstacle and its shape.
- The execution of some instructions may take more time than others (for example instructions to write on LCD, serial monitor or SD card).
- The detection of some obstacles may require the use of more instructions than others.

To obtain a synchronous system as much as possible, I created a scheduler by using a simple *"if ()"* statement that executes all the code related to the program logic only if a certain time ΔT has elapsed.
Inside the *"loop ()"* function, I used the *"millis ()"* instruction to calculate the time elapsed between the instant t_2 when the code execution begins and the instant t_1 when it ends (and have to start over).

```
void loop () {

t1 = millis ();

  if (t1-t2 >= dT) {

    t2 = millis ();
    ...      // code execution
  }
}
```

If the difference t_1 - t_2 is equal to or greater than the ΔT value, then a new cycle can be performed.
To calculate the correct Δt time value, I computed the average execution time of the code and the average number of measurements performed by the sensor.
Using the serial monitor, I obtained the average number of measurements per second, equal to about 45 measurements/s.

$$\Delta T_{average} = \frac{1000\ ms}{45\ measurements} = 22{,}2\ ms$$

The number of measurements was initially very low due to the numerous instructions used to write on the LCD display and to save data on the SD memory. Consequently, the average cycle time $\Delta T_{average}$ obtained was very high.
Therefore, it was necessary to optimize the written code for logic programming in order to speed up the execution of a cycle time and increase the number of measurements. To solve the problem related to writing on the LCD I decided to use a refresh counter *"refresh_cnt"* in order to write the measurements on the screen only once in 10.

```
if (refresh_cnt == 0) {

    lcd.setCursor(0, 0);
    lcd.print(distAvg);
}

refresh_cnt++;
refresh_cnt = refresh_cnt % 10;
```

A similar strategy has been adopted for saving data on the SD memory. Opening the file and saving data to it is a much more time-consuming operation than simply writing to the display.
To avoid opening the file every time a new measurement is received, I resorted to the use of two *"String"* type variables and a *"SD_cnt"* counter.

```
sdData = String (distAvg);
sdData.concat(", ");
SDtemp = String(flag);
sdData.concat(SDtemp);
```

Each time a measurement is received, it is concatenated into a string thanks to the use of a temporal variable ("SDtemp") and the *"concat ()"* function. With each measurement received, the counter is incremented and only when it reaches the maximum value, the file is opened and the value of the entire string is written.

```
if (SD_cnt == 39)
{
    ...    // open file and store the string
}

SD_cnt++;
SD_cnt = SD_cnt % 40;
```

It was not possible to create a string of more than 40 measurements as the size (in terms of bytes) of the buffer used for serial communication was exceeded. For each measurement, the value of the *"flag"* variable was also saved, in order to understand if the logic used for the obstacle detection was correct. Therefore, the counter number was set to 40.
After making these optimizations, using the serial monitor and doing the computations, the average number of measurements made by the sensor per second has risen to 130. This value is much better, in fact, if you calculate the average time between two consecutive measurements (cycle time), you get:

$$\Delta T_{average} = \frac{1000\ ms}{130\ measurements} = 7{,}7\ ms$$

Subsequently, using the *"millis ()"* instruction, I calculated, the minimum cycle execution time to verify that the data were coherent. As shown in Figure 4.5 below, I got $\Delta T_{min} = 7\ ms$.

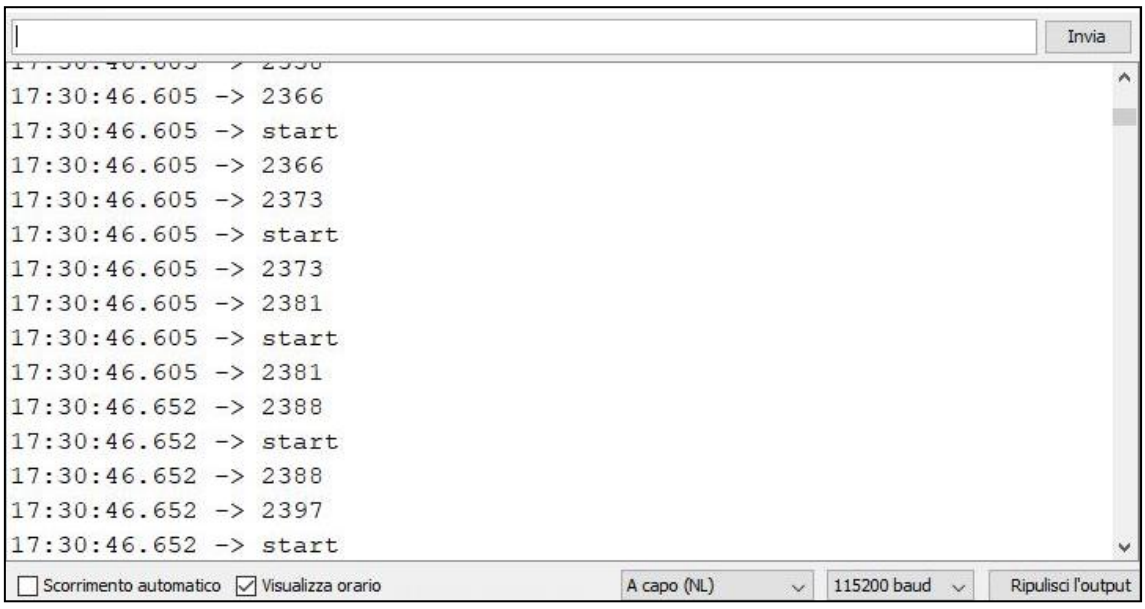


Fig. 4.5: Detection of the minimum cycle execution time

The average time, however, cannot represent the correct value of the scheduler as it is not constant. To have a constant value, I proceeded by step by increasing the value of ΔT until the execution time ΔT reached a constant value.
With $\Delta T = 8\,ms$ I was able to reach the goal but for greater safety, I decided to maintain a certain margin by setting $\Delta T = 10$ ms.
In this way, the sensor takes a measurement precisely every 10 milliseconds for a total of 100 measurements per second.
By dividing by the vehicle speed, it is possible to obtain the number of measurements made for each meter traveled:

- $$N_{@10\text{ km/h}} = \frac{100\ {measuremts}/{s}}{2{,}77\ m/s} = 36\ {measurements}/{m}$$

$$d_{@10\text{ km/h}} = \frac{100\ cm}{36\ {measuremts}/{m}} \approx 2{,}8\ cm$$

- $$N_{@25\text{ km/h}} = \frac{100\ {measuremts}/{s}}{6{,}94\ m/s} = 14\ {measurements}/{m}$$

$$d_{@25\text{ km/h}} = \frac{100\ cm}{14\ {measurements}/{m}} \approx 7{,}2\ cm$$

According to the calculations, considering a speed of 10 km/h (average speed of a bicycle), the sensor is able to perform 36 measurements every meter; therefore, it is able to take a measurement in less than 3 cm. Instead, considering a speed of 25 km/h (maximum speed allowed), the sensor is able to make 14 measurements, that is one measurement every 7,2 centimeters.
As the vehicle speed increases, the number of measurements, that the sensor performs every meter, decreases and, therefore, the probability of correctly detecting an obstacle decreases. At maximum speed, the sensor may not detect small obstacles.
The speed factor is very important in detecting obstacles as it allows us to adapt the acoustic signal to the speed with which the driver risks hitting the obstacle.
In my project, I have chosen that the sensor operates only for speeds equal to or greater than 5 km/h for two reasons:

1. On average, two-wheeled electric vehicles already exceed 5 km/h with the first push you give to get moving. Therefore, it is difficult to go at a lower speed if you consider that the average speed of an adult person is 5 km/h.

2. If the speed is lower than 5 km/h it means that the driver is either braking or cannot accelerate as if there are many obstacles or people close in front of him. This could lead to a continuous beeping of the sensor due to the presence of very close obstacles and it would be very unpleasant to hear the repeated sound of the buzzer incessantly.

In light of these considerations, I have chosen to disable all the sensor logic if the speed is lower than 5 km/h.

```
void obstacle (int distance, long distAvg) {

    if (Speed == 5) {

        ... // all logic with flag evaluation
    }
}
```

4.5.1 Logic programming

Each obstacle requires a different type of logic to be developed and, therefore, different instructions to be executed based on the obstacle type. At the software level, all the logic used for obstacle detection has been incorporated into the *"obstacle ()"* function.

```
obstacle (dist, distAvg);
```

This function receives in real time the distance measurement *"dist"* carried out by the sensor and the temporary average value of the measurements *"distAvg"*. Through these two data, it performs all the necessary computations to detect the presence of an obstacle and discriminate the type of obstacle.
The first thing the function does is understand what kind of obstacle the sensor is detecting.
To discriminate the different cases, I used a *"flag"* variable which is set with different values depending on the obstacle. Table 4.5.1 shows the different values that the variable can assume.
The *"flag"* variable is set to the default value '0' and will remain so if the sensor does not detect any obstacles. If, on the other hand, the measurements made by the sensor begin to deviate from the value of 10 meter, the *"flag"* variable will be set thanks to the use of conditional instructions *"if/else ()"*.
Figure 4.5.1 shows the graph of the finite state machine (FSM), which allows you to better understand what happens in the *"obstacle ()"* function.

Flag	Obstacle Type
4	Moving Obstacle
3	Generic Stationary Obstacle
2	Step Up
1	Road Bump
0	No obstacle
-1	Ditch
-2	Step Down
-3	Discent / Ravine

Tab. 4.5.1: Flag table

The state machine shows how obstacle detection occurs from a logical point of view. At the beginning, the sensor starts from the "No obstacle" state, which we can consider as the default state.
Whenever the sensor detects no obstacles, it will be in that state. From the default state, there are four possible cases:

1. The measured distance starts to increase.
2. The measured distance starts to decrease.
3. The measured distance increases radically.
4. The measured distance decreases radically.

From the code logic point of view, if the distance starts to increase, the case of the ditch will be set immediately (flag = -1), vice versa, if the distance starts to decrease, the case of the road bump will be set (flag = 1). Furthermore, these two states are intermediate states for other states since, as soon as a change in the pointing distance is detected, the sensor is not able to immediately understand the type of obstacle in front of it and it will start by setting either the case *"ditch"* (greater distance) or the *"bump"* case (shorter distance). The sensor will be able to understand the type of obstacle only by noting the occurrence of different conditions through the succession of measurements.
For example, if the measured distance decreases there may be more possible cases: it could be a road bump, a step or a generic obstacle with greater size.
Then, the sensor will scan the type of obstacle through an algorithm and a series of conditional statements *"if/else ()"* and based on the type it will set another state.
The only types of obstacles that can be immediately discriminated are moving obstacles and descents (or ravines), since in these two cases there is a clear and conspicuous change from the 10 meters distance to the new measured distance. For

example, if the distance changes rapidly from 10 meters to 30 meters, it means that there is a ravine of 2 meters or a very steep descent.

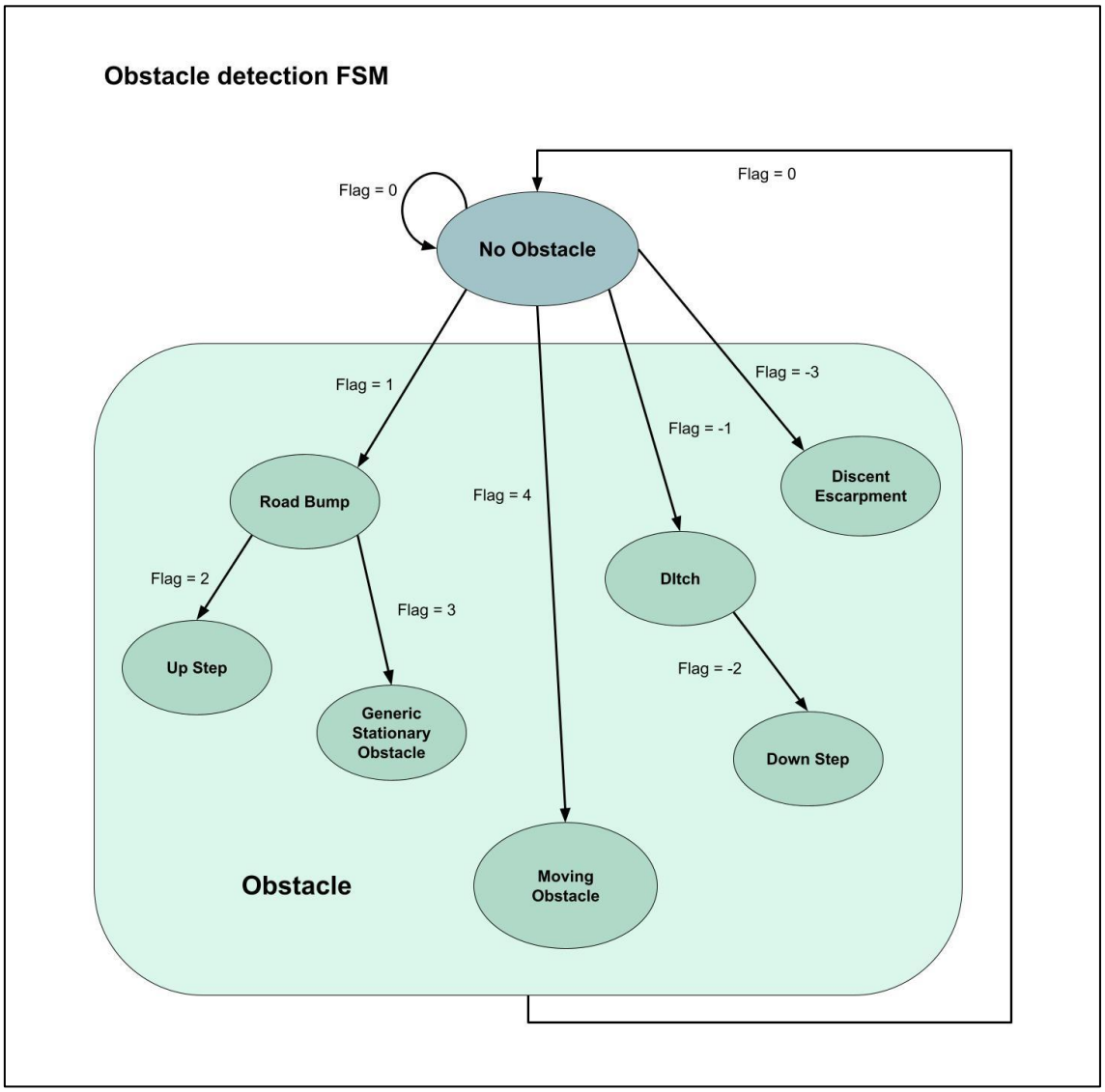


Fig. 4.5.1: Obstacle type detection FSM

4.5.2 Ditch/Bump detection

If we consider a sensor with a constant pointing distance (10 m), detecting obstacles on the roadway can be translated into measuring the difference between the ideal value and the measurements made by the sensor. If, for example, there is a ditch on the road, the distance detected by the sensor should increase for a certain amount of time, before returning to the ideal value of 10 meters. Similarly, if there is a road bump, step or any other object on the roadway, the detected distance should decrease more and more until the sensor laser beam has passed that obstacle.

By calculating the variation of the distance pointed by the LiDAR, it is also possible to mathematically obtain the depth or height of the obstacle.

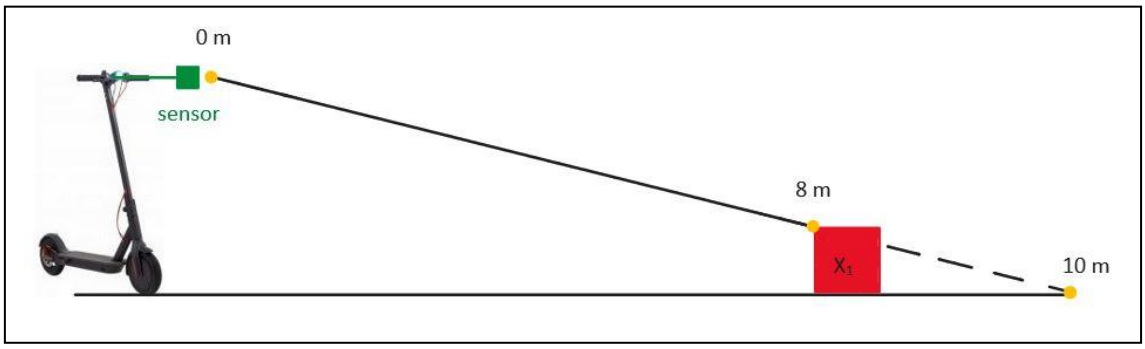


Fig. 4.5.2a: Obstacle height measurement

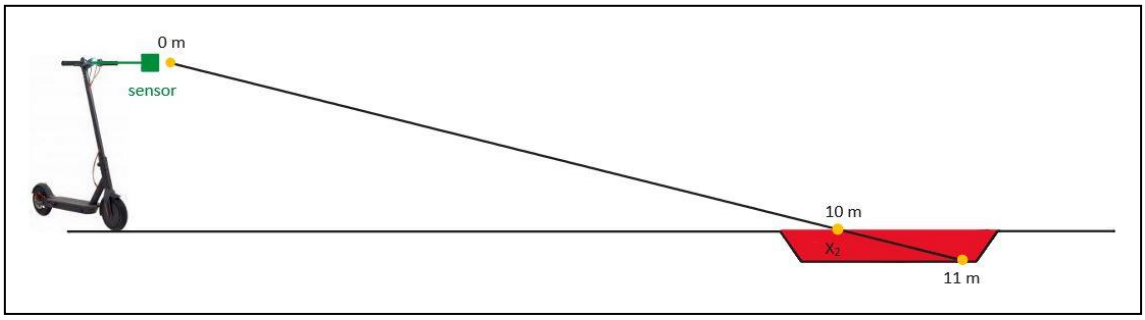


Fig. 4.5.2b: Ditch depth measurement

If you consider that between the sensor height and the distance pointed by it there is a ratio of 1:10, you can easily calculate the size of the obstacle by dividing the maximum variation of the measurement from the pointed distance by 10.
If, for example, an obstacle x_1 is interposed in front of the vehicle and breaks the LiDAR beam up to a minimum distance of 8 meters, the height of the obstacle will be computed as follows:

$$height_{x_1} = \frac{10\,m - 8\,m}{10} = 0.2\,m = 20\,cm$$

If, on the other hand, the sensor beam crosses a ditch and the measured distance increases, reaching, for example, a maximum value of 11 meters, the ditch depth x_2 will be calculated as follows:

$$depth_{x_2} = \frac{11\,m - 10\,m}{10} = 0.1\,m = 10\,cm$$

4.5.3 Ditch/bump code description

Two-wheeled vehicles, as mentioned in the previous paragraphs, are subject to various oscillations due to their instability; therefore, it was necessary to include a tolerance range to prevent these oscillations from causing the presence of false obstacles.

Therefore, I have set a threshold value *MIN_VAR* which allows us to be sure that the sensor is detecting a real obstacle. *POINT_DIST*, on the other hand, represents the pointing distance and is also a constant set to the fixed value of 10 meters. If the difference between the average distance "*distAvg"* and the constant *POINT_DIST* exceeds the threshold value *MIN_VAR* the variable *"flag"* is set to the corresponding value.

```
if (flag == 0) {

    If (distAvg - POINT_DIST > MIN_VAR) {

        flag = -1;    // Ditch/StepDown/Discent

    } else if (POINT_DIST - distAvg > MIN_VAR) {

        flag = 1;     // roadBump/StepUp/genericObstacle

    }
}
```

The threshold value can be changed, but a compromise is needed because if the value is too low, we can risk to detect false obstacles while, if it is too high, we can omit the detection of some small obstacles.

The Lidar-Lite v3 sensor, for distances greater than 5 m, has an accuracy of ± 10 cm (as reported in Table 3.1d). If we take into account the 1:10 ratio between sensor height and laser beam diagonal, an accuracy of ± 10 cm corresponds to an obstacle height accuracy of ± 1 cm. However, the "signal strength" factor described in paragraph 4.3.2 have also to be taken into consideration; at a distance of 10 meters, we have a beam diameter of 10 cm (dist/100) and it is all the more difficult to detect obstacles the smaller they are compared to the beam diameter.

The *MIN_VAR* value was initially set at 50 cm (with a corresponding height accuracy of ± 2 cm) but, in the test phase, it will be necessary to verify the effectiveness of this value or to modify it in order to find the right compromise. After having performed the discrimination, based on the value of the *"flag"* variable, a specific piece of code is then executed to detect the obstacle height or depth peak where, the *"ditch"* and *"bump"* variables are used to keep trace of the minimum and maximum value respectively. The structure of the code is the same for both types of obstacles but with opposite logic. The following is the code relating to the ditch case to understand how the code is structured.

Once you have entered the specific case of the ditch, there are only two possibilities:

1. If the difference between the distance *"dist"* and the constant *POINT_DIST* exceeds the threshold value *MIN_VAR* it means that the sensor has started measuring an obstacle. From when the ditch detection begins, the measurements performed are compared with the minimum *"ditch"* value and if a new peak value is detected, the variable is updated to that value.

2. If, otherwise, this difference is less than the threshold value, it means that the laser beam has passed the obstacle and is returning to detect the theoretical 10 meters. At this point, it is possible to output the peak value found corresponding to the ditch depth.

To exit the *"ditch"* case, the value of the *"flag"* must be reset to '0' in order to return to the initial condition from which, then it is possible to search for a new obstacle.
It is therefore clear that once the discrimination has been performed and the flag has been set to the corresponding value, it will not be possible to enter in another obstacle case state if the detection of the current obstacle is not completed.
To reset the *"flag"* and all other variables used, I exploited the *"End"* variable. The latter is set to "true" at the end of the obstacle detection. If *"End"* is true all the variables used in the obstacle case are reset and you return to the initial state.
The code relating to the road bump case has not been reported, but has a logical structure equal and opposite respect to the ditch case.

DITCH code

```
if (flag == -1) {

  if (distance - POINT_DIST > MIN_VAR) {

  // Ditch still detected
    if (distance > ditch) {
      ditch = distance;
    }

  } else if (distance - POINT_DIST <= MIN_VAR) {

  // End of the ditch
      lcd.print ((ditch-POINT_DIST)/10);
      lcd.print (" cm ");
      End = true;
  }
}
```

4.5.4 Possible problems in depth computation

The height or depth computation of the obstacle is performed by searching for the maximum or minimum value in the range of measurements executed on the obstacle. In the case in which the sensor detects the ditch depth there is, however, the possibility that the calculated depth is not correct. In fact, it must be taken into account that the sensor laser beam points to the ground transversely and it could happen that the sensor is unable to perform measurements on the entire surface of the ditch, but only on a part of it (the final part).

If we consider, for example, the ditch represented in Figure 4.5.4, it is possible to notice that there is a blind area (highlighted in green) that the laser beam will never reach. The laser sensor is unable to obtain the measurement relative to the true depth of the ditch since the range of measurements performed on the ditch does not map its entire area but only a part.

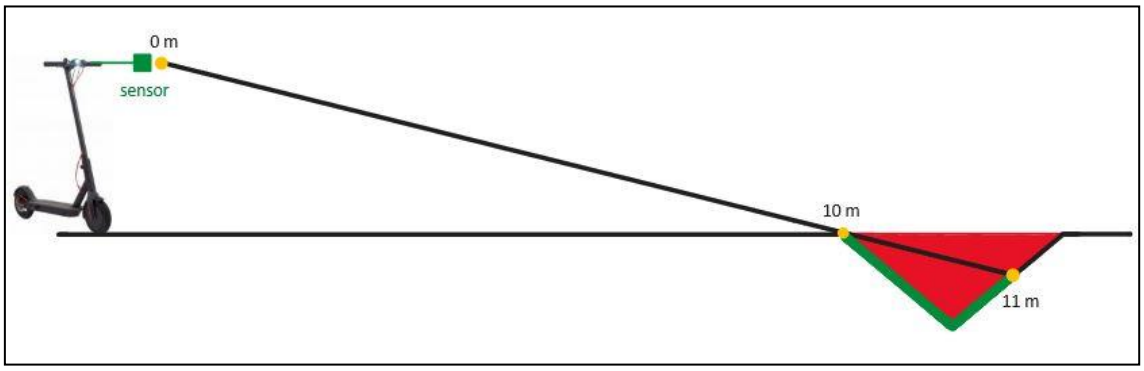


Fig. 4.5.4: Depth computation problem

In the case showed in figure, the sensor will detect a depth of 10 cm (distance change of 1 metro), but in reality, the ditch is deeper and the real depth will not be detected by the sensor.

This can happen when the ditches are narrow and deep; more precisely, considering a line obtained by joining the point where the ditch begins with its deepest point, if the inclination of this line (respect to the line) is greater than that of the laser beam, it will not be possible to detect its true depth.

4.5.5 Step detection

The most classic obstacle you can run into when driving a two-wheeled vehicle is the sidewalk. The latter is one of the most frequent causes of accidents.

The detection of a step was conceived as an extension of the ditch/bump case. In fact, assuming that the sensor laser beam is going to point a step, the taken measurements should deviate from the 10 meters value, decreasing or increasing until they reach a constant value. The sensor, therefore, will initially recognize a bump or ditch and only at the end, when the sensor will realize that it is receiving constant peak measurements, it will discriminate the obstacle as a step.

Figure 4.5.6a shows the case in which the sensor detects a “up step” S_1.

The measurements made by the sensor will decrease until they reach a constant value (highlighted in yellow) as the laser beam points to a plane with greater height.

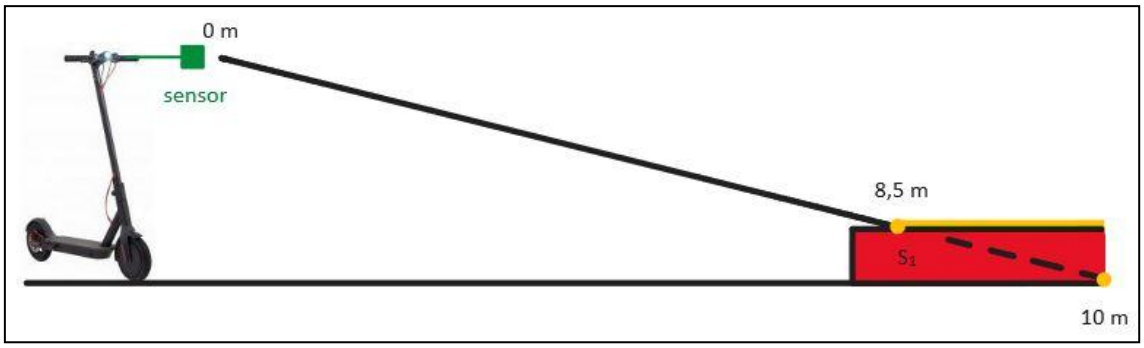


Fig. 4.5.5a: Up step measurement

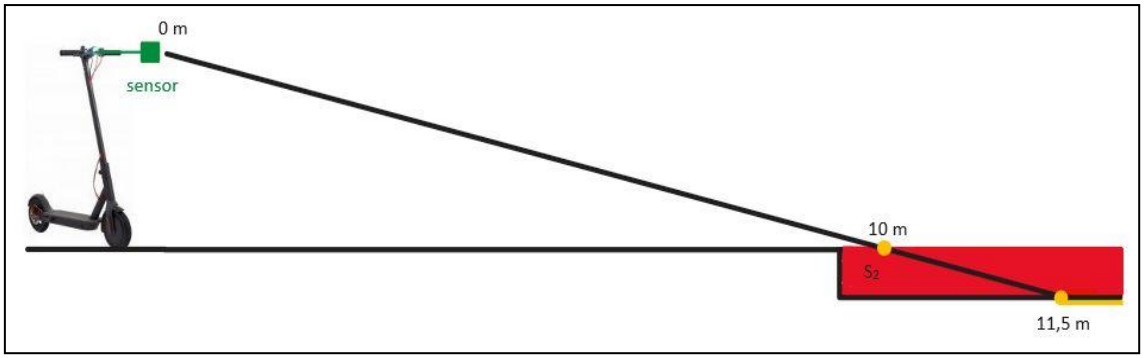


Fig. 4.5.5b: Down step measurement

The same thing happens in the opposite case. When the sensor beam is going to point a "down step" (Figure 4.5.6b), the value of the measurements increases until it reaches a new constant value as the beam detects to a difference in plane height.
By repeating the calculations made in the previous paragraph, it is possible to detect the height of the steps, in the same way, by computing the difference between the measurements of the two planes and dividing by 10.

$$height_{S_1} = \frac{10\,m - 8{,}5\,m}{10} = 0.15\,m = 15\,cm$$

$$depth_{S_2} = \frac{11{,}5\,m - 10\,m}{10} = 0.15\,m = 15\,cm$$

It is obvious that, if we consider the ascent and descent from the same step, the height value detected by the sensor, in both cases, must be the same.
To implement the logic that allows to detect a step it is necessary to use a time variable. In fact, the only way to verify that a measurement remains constant is to

check its value over time. To do so, it is necessary to verify, after a certain movement d_x, that the value of the average distance returns to be constant. But, the time t_x it takes to execute this movement varies according to the vehicle speed. Practically:

$$t_x = \frac{d_x}{v_{vehicle}}$$

where, $v_{vehicle}$ is the vehicle speed.
To choose the value of d_x it was necessary to find the right compromise since, if it is too high it can lead to latency in the detection of the obstacle, if too low the *"distAvg"* value may not have had enough time to stabilize. The value of the d_x movement has been initially set to 50 centimeters.
Each time a cycle is performed in the *"loop ()"* function, a *"cycle_cnt"* counter is incremented and knowing the cycle time value, calculated previously, it is possible to have a time reference. As soon as the sensor begins detecting the obstacle, the value of *"cycle_cnt"* is copied to the *"start_mis"* variable. By multiplying the difference between the values of these two variables by the cycle time ΔT, it is possible to obtain the time elapsed from the start of the obstacle detection.

$$elapsed\ time = (cycle\_cnt\ - start\_mis) * \Delta T$$

If the elapsed time is greater than t_x and the average distance *"distAvg"* remains constant, the obstacle is recognized as a step.

Note:
To obtain the time t_x, the vehicle speed must also be known. In most electric two-wheeled vehicles there is a sensor that detects the speed of the vehicle or, in any case, this parameter is calculated thanks to the position data obtained from the GPS in real time. Since there is no sensor available to detect the $v_{vehicle}$ speed, in the test phase some speed values have been set manually to check the correct operation of the sensor.

DOWN STEP code

```
if ((cycle_cnt - start_mis >= tx) &&
      (abs (distance - distAvg) <= MIN_VAR)) {

    if ((ditch - POINT_DIST <= STEP_MAX)) {
        flag = -2;        // down Step

    }
}
```

UP STEP code

```
if ((cycle_cnt - start_mis >= tx) &&
      (abs (distance - distAvg) <= MIN_VAR)) {

    if ((POINT_DIST - bump <= STEP_MAX)) {
        flag = 2;        // up Step

    }
}
```

4.5.6 Steep descent or escarpment detection

Very steep descents, escarpments or, even worse, ravines are obstacles that can seriously endanger the life of a two-wheeler driver. The detection of this type of obstacle is very simple to perform as if you have a large escarpment or a steep descent the distance measured by the sensor will increase considerably.
If there is a slope with a more inclined plane than the laser beam, the measurements made by the sensor will not gradually increase since, as described in the previous paragraph, there will be a hidden area that the beam will never be able to point. What the sensor is able to target is the plane next to the grading as shown in the following Figure 4.5.6.

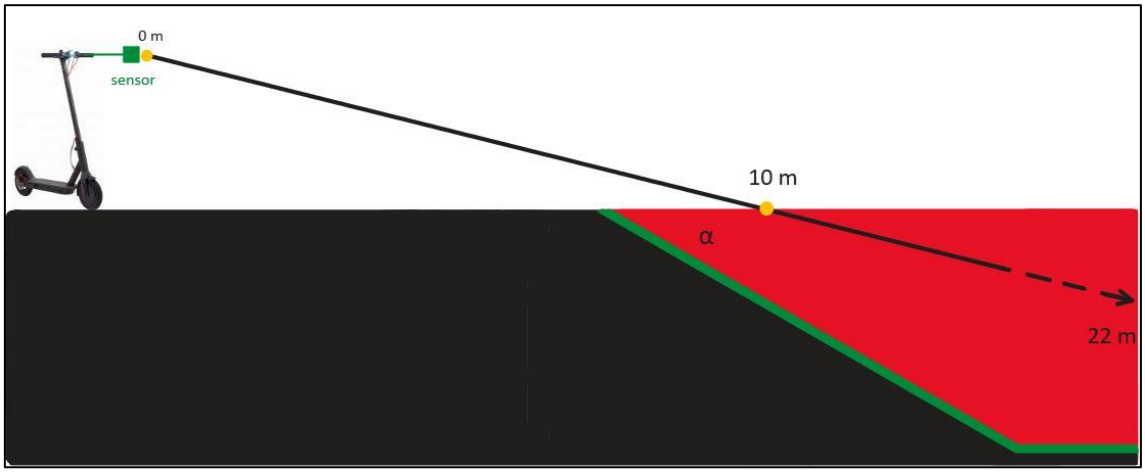


Fig. 4.5.6: Escarpment depth measurement

In the figure you can clearly see how the measured distance changes sharply from the 10 meters standard distance to reach 22 meters on the next plane. In this case the sensor will detect a depth equal to the difference in height of the two planes.

$$depth = \frac{22\,m - 10\,m}{10} = 1{,}2\,m$$

It should be noted that, in this case, it is not possible to distinguish the type of obstacle in front of us. Due to the blind area that cannot be detected by the sensor, the latter is able to detect the depth but cannot understand if there is an escarpment or a descent in front of it. In fact, all the area highlighted in green is hidden and the inclination angle "α" of the plane could be either a few degrees or 90°. Therefore, I will only limit myself to generically indicating the type of obstacle, showing its depth.
To detect this type of obstacle, at the software level, I simply used the *"if ()"* conditional statement. If the distance value suddenly increases by a value greater than 10 meters (corresponding to a difference in height of 1 meter), the value of the *"flag"* variable is set to "-3".

```
if (distAvg - POINT_DIST > 1000) {
    flag = -3;
}
```

Once the type of obstacle has been recognized, the depth value is continuously updated and output on the display. Then the sound signal is also activated through the *"alert ()"* function with a frequency that varies progressively according to the value of the speed variable *"Speed"* and the time elapsed from the detection of the obstacle.

ESCARPMENT code

```
if (flag == -3) {

    if (distance > ditch) {
        ditch = distance;
    }
    //display alert
    // sound alert
}
```

4.5.7 Generic stationary obstacle detection

The detection of a generic obstacle is done as an extension of the bump case. The LiDAR sensor, in fact, initially recognizes the measurement change but is not yet able to understand if the obstacle is just a road bump or is a much bigger obstacle. As shown in the FSM in Figure 4.4, as soon as the sensor detects a distance decrease, it will initially set the "bump" case and only later, if certain conditions are matched, it will set the "generic obstacle" case.

“What are these conditions?”

If the measured distances exceed a certain *BUMP_MAX* threshold, then the obstacle is classified as a "generic obstacle" and not as a “bump”.
Road bumps usually reach a maximum height of 10-15 centimeters, therefore, *BUMP_MAX* has been set to a value of 150 centimeters (which corresponds to a height detection of 15 cm).

```
if (POINT_DIST - bump > BUMP_MAX) {
    flag=3;
}
```

Once the obstacle is detected, you enter a cycle similar to that of the “bump” but, here, in addition, the distance from the obstacle is also monitored over time. In fact, we could have a big stationary obstacle, higher than the sensor, and the measured distance would continue to decrease without ever returning to point to the ground. Therefore, as the distance decreases, a beep signal is emitted with increasing frequency.
If the driver does not deviate the obstacle, the warning signal will continue until the possible impact with the obstacle. If, on the other hand, the driver does not deviate the obstacle but brakes, the vehicle speed is reduced and if the speed falls below the 5 km/h threshold the sensor will not work. As mentioned in paragraph 4.5, the sensor will be activated only if a certain threshold speed is exceeded, therefore, if the driver brakes o slows down, the warning signal ends because the driver has noticed the danger and it is not necessary to have a harassing sound.
Unfortunately, the sensor prototype created is unable to distinguish obstacles higher than the sensor, as shown in figure 4.5.7. If the laser beam is interrupted at a distance of 5 meters, for example, the sensor cannot distinguish whether it is a person, a wall or an ascent.
The sensor emits a single beam which, although it is not point-like (the diameter is about 10 centimeters at a distance of 10 meters), it can only detect obstacles that are in the direction in which it is pointed.

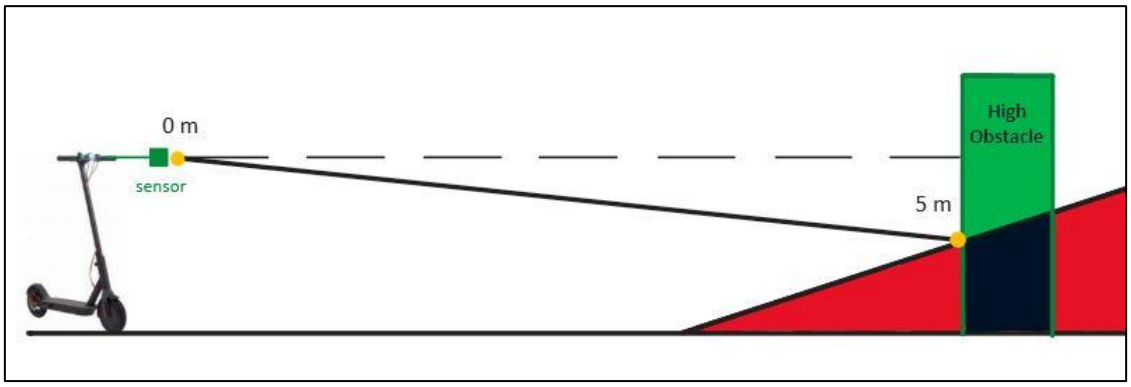


Fig. 4.5.7: Generic stationary obstacle detection

Furthermore, since the height of the sensor is 1 meter, it is not possible to detect the size of very tall objects. Since the sensor is able to calculate the height of obstacles based on how much the measured distance varies, it follows that the higher the obstacles are, the closer they will be. This is why I decided to only detect the height of obstacles up to a maximum of 50 centimeters, while in all other cases the sensor notifies the obstacle only based on its proximity in order to accentuate the danger.

GENERIC STATIONARY OBSTACLE code

```
if (flag == 3) {      // generic stationary obstacle

    ... //same operations as bump

    if (((cycle_cnt - start_mis) > (100/(Speed/3.6))) &&
         (POINT_DIST - distAvg > BUMP_MAX)) {

      if (refresh_cnt == 0) {
        // alert on display
        // alert with buzzer
      }
    }
}
```

4.5.8 Moving obstacle detection

The detection of a moving obstacle differs from the case of a stationary obstacle and can be recognized quickly. By moving obstacle, we mean both an obstacle that cross the driver trajectory without stopping and an obstacle that suddenly places itself in the vehicle trajectory.
When an obstacle is moving and cuts the laser beam, the distance measured by the sensor decreases sharply and then returns to the standard 10 meters value.

```
if ((POINT_DIST - distAvg > MIN_VAR) &&
     (distAvg - dist > STEP_MAX)) { // Moving Obstacle

    flag = 4;
}
```

Therefore, with a sharp decrease of the distance, the *"flag"* is set to the value '4'. After that, the sensor immediately signals the passage of the obstacle and monitors its distance if it continues to remain in the path of the vehicle, signaling to the driver the proximity of the obstacle.

MOVING OBSTACLE code

```
if (flag == 4) {     // Moving obstacle

    if (POINT_DIST - distAvg > MIN_VAR) {

        if (distAvg < hump) {
          hump = distAvg;
        }

        if (refresh_cnt == 0) {
            // alert on display
            // alert with buzzer
        }

    } else {
        End=true;
    }

}
```

Chapter 5

Prototype tests

The test phase is a very important and decisive phase as it allows us to understand the feasibility of our POC. In the last part of my thesis project, I performed tests on the sensor prototype to analyze its data and verify its correct functioning. In particular, it is necessary to check if the parameters chosen during the programming of the device are effective or if I need to make changes to optimize the operation of the sensor.
Testing a prototype is not an easy task and, therefore, I decided to proceed step by step, starting from the simple distinction between obstacle and free road and then gradually adding other features, up to testing the sensor in its complete operation. In this way it is also easier to identify errors and malfunctions of the device, starting from the most generic cases up to the most specific ones.

5.1 Prototype installation

The installation of the sensor, besides being the first part of this phase, is certainly the most relevant one. In fact, if the sensor is not fixed correctly, there is the risk of compromising the accuracy of the measurements performed and, therefore, the operation of the sensor.
In order to perform the tests, it was necessary to mount and fix the sensor prototype on a two-wheeled vehicle and, in my case, I used a mountain bike.
It was not easy to place the prototype due to the particular geometry of the bicycle. The LiDAR sensor needed a very strong and stable anchor; therefore, it was fixed to the center of the handlebar using an iron frame and plastic cable ties.
The iron support structure was created by hand, using two corner plates and some screws. I created a structure that was as rigid and resistant as possible, in order to keep the sensor motionless if the vehicle was subjected to particularly strong vibrations or blows (such as colliding with a road bump).
Referring back to the study carried out in chapter 1, the sensor was placed at a 1-meter height, tilting the sensor to obtain a pointing distance that was as close as possible to the theoretical 10 meters.
Below there are some photos showing how the sensor prototype was fixed to the vehicle. In particular, in figure 5.1b (side view) it is possible to notice the small inclination angle of the sensor. With a just 5 ° angle (approximately) it almost seems as if the laser beam has a direction parallel to the ground. Due to vehicle vibrations,

there is a risk that the distance measured by the sensor may change and alter the correct functioning of the device, therefore, it will be important to analyze the behavior of the vehicle on road surfaces with different conformations.

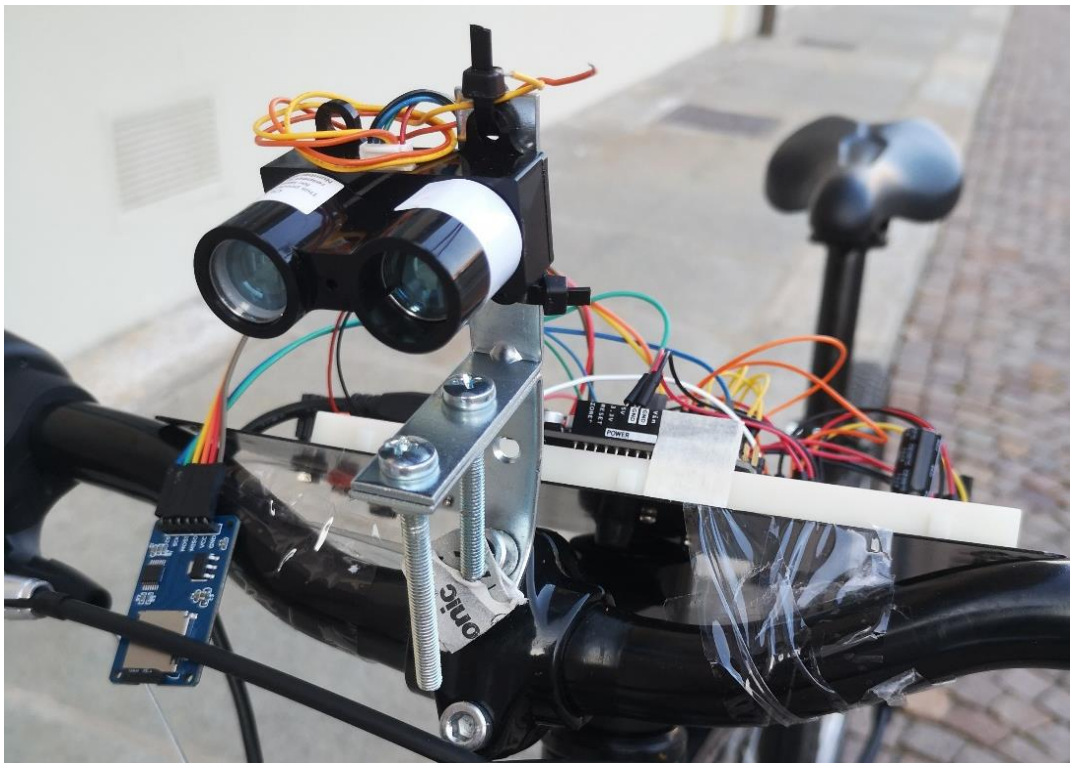


Photo 5.1a: Structure used for Lidar sensor fixing

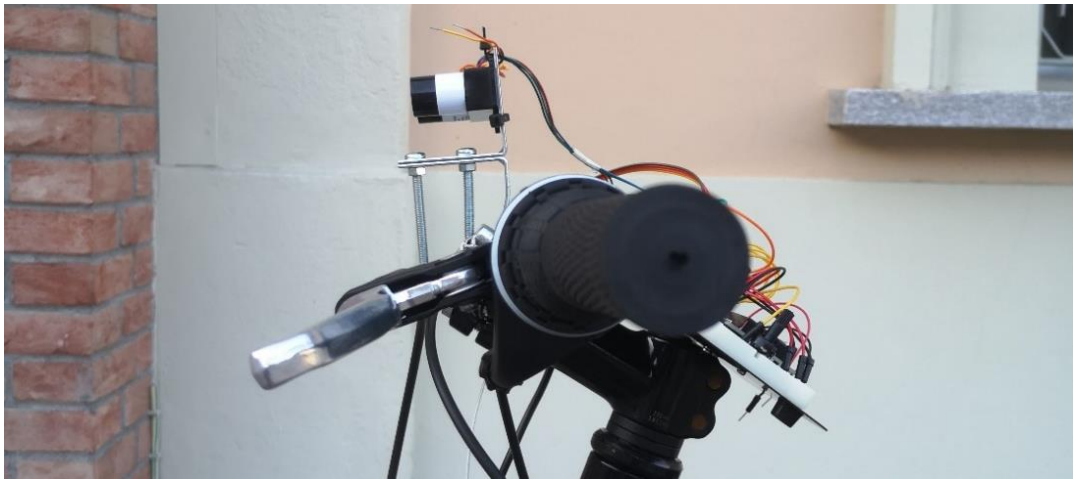


Photo 5.1b: Sensor tilt (side view)

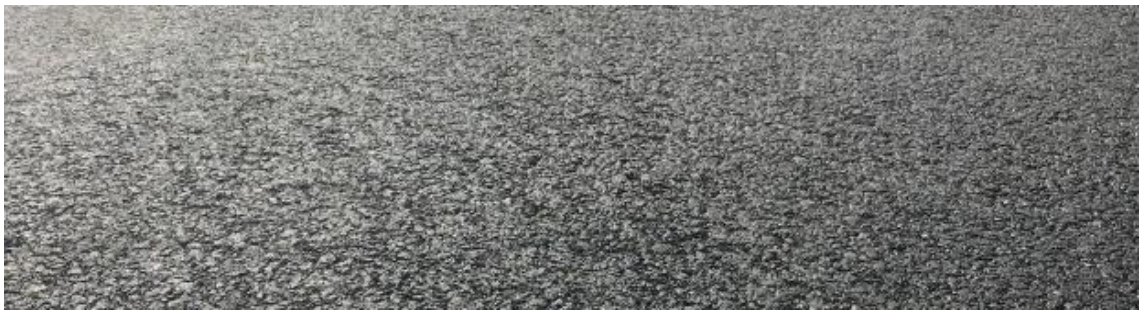
The breadboard, which contains the Elegoo UNO microcontroller, the LCD display, the buzzer and the SD reader, was placed on the bike handlebar while the power bank was fixed on the top of the frame for added convenience. Both were fixed with transparent polypropylene adhesive tape. Photo 5.1c shows the complete structure of the prototype used for the entire test phase.

Photo 5.1c: Complete prototype structure

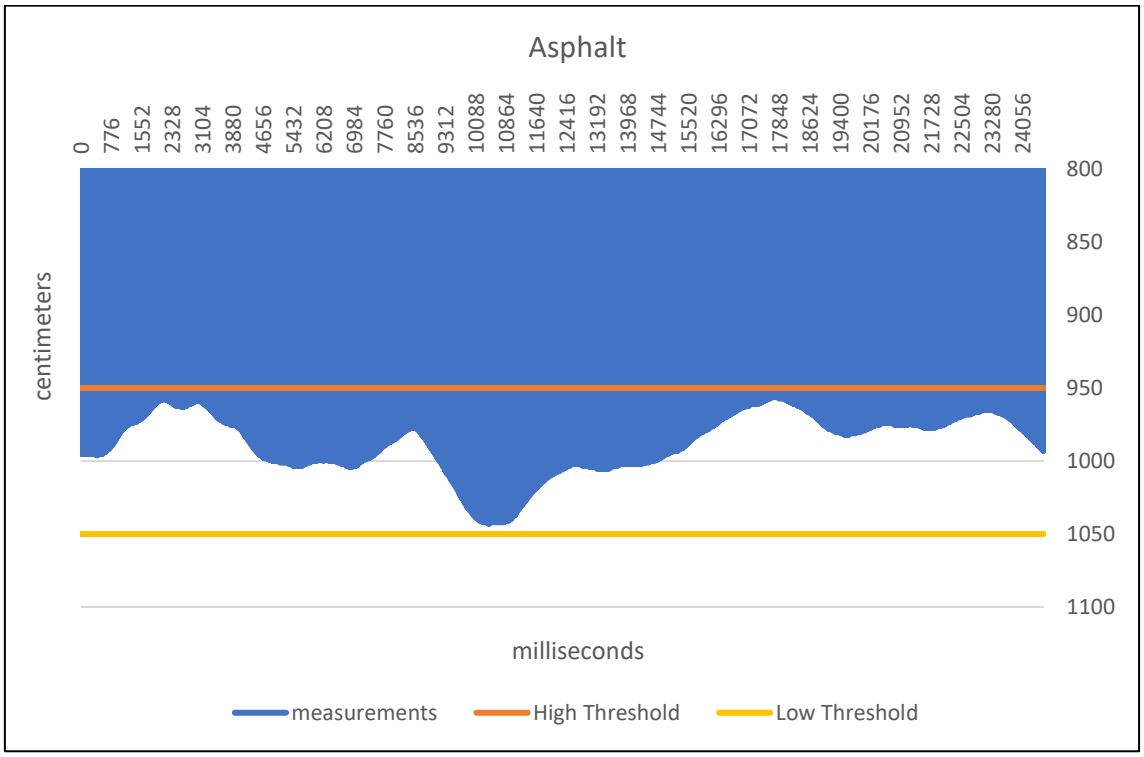
5.2 Phase I: Free road measurements check

After placing and fixing the sensor prototype on the bicycle and creating my test vehicle, the first test phase on the device began.
Initially, a very simplified version of the complete code was loaded in order to verify, first and foremost, the basic behavior of the sensor during a ride. The first version of the code allows to discriminate only the presence of an obstacle without performing any recognition logic. In this regard, I started the test phase by analyzing the accuracy of the measurements made by the sensor to check if the thresholds and parameters set during programming were correct.
The oscillations and vibrations to which the vehicle is subjected during the ride can affect the accuracy of the measurements, compromising the obstacles recognition. To understand how the different road surface conformations could influence the behavior of the device, I performed tests on different types of terrain and road pavements and, subsequently, I analyzed the data provided by the sensor. Using an internal courtyard of the Teoresi company, I tested the sensor's behavior on three road types: dirt road, pavé (cobblestones) and asphalt road.
Since I used a non-electric bicycle, I could not set a cruising speed of 10 km/h and, therefore, I tried to manually maintain a similar constant average speed, checking with a bicycle speedometer.

5.2.1 Asphalt road



The asphalt covers most of the urban roads and is the place where the vehicles usually move. First of all, therefore, I ran test on a common asphalted city road. From the data, I would have expected an almost low margin of fluctuations as asphalt roads usually have a certain homogeneity, as the roads are flattened.
Graph 5.2.1 shows the trend of the measurements performed by the sensor during a short ride on the asphalt. The graph shows the distances measured (in centimeters) by the sensor on the ordinate axis, while on the abscissa the time axis that is shown in milliseconds. The ordinate axis was voluntarily overturned in order to have a clearer representation.
From the graph it is possible to see how the asphalt road surface, although it may seem smooth and homogeneous, still presents some ripple due to the soil geomorphology. For this reason, the measurements taken outside differ significantly from those carried out internally.



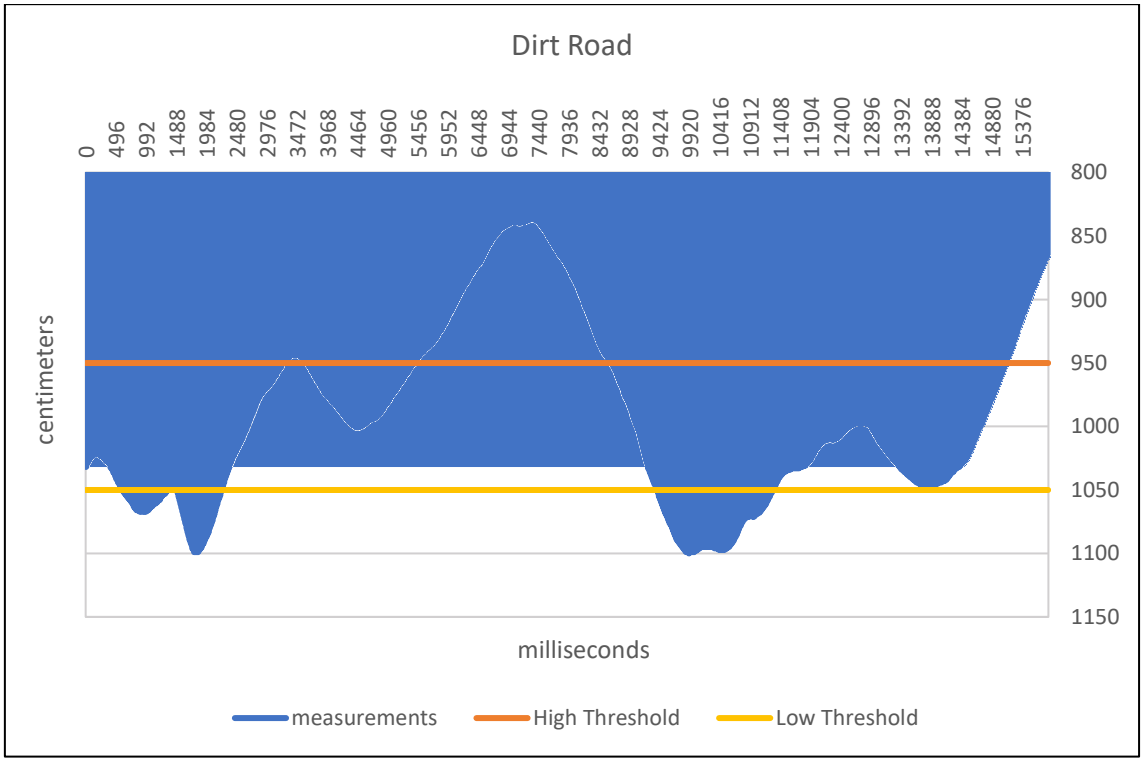
Graph 5.2.1: Measurements on the asphalt road

The trend of the measurements is therefore very different from the one supposed. First of all, it is necessary to modify the value of the thresholds so that the sensor does not detect false obstacles due to the oscillations to which the vehicle is subjected.
The "MIN_VAR" value, initially set at 20 centimeters, was thus increased to 50 centimeters. This means that, now, the sensor will not be able to discriminate obstacles with a height lower than 5 centimeters.

5.2.2 Dirt road

Subsequently, I performed tests on a dirt road. What I expected was a worse behavior than the one obtained in the asphalt case. The nature of the ground, extremely uneven and full of breaches, should have led to numerous oscillations of the vehicle, greatly influencing the value of the measurements.

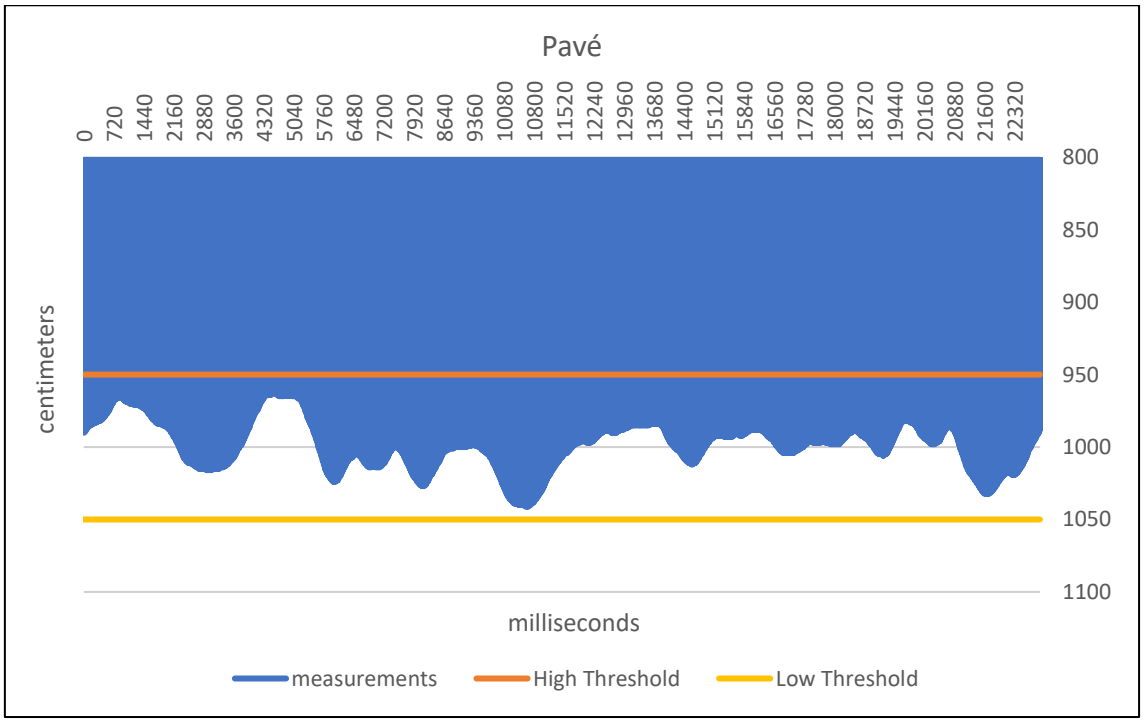
The data relating to the measurements taken on the dirt road are shown in graph 5.2.2.
As expected, the measurements taken undergo a large variation due to vehicle oscillations. Although the terrain is uneven, and therefore, it is right to think of obtaining a road layout full of roughness, the measurements obtained far exceed the threshold set of 50 centimeters, even reaching peaks of 150 centimeters (i.e., 15 cm in height).
The cause of this is not only the nature of the ground but also the presence of the shock absorbers which on the one hand favor a softer ride, but, on the other hand, create a wavy movement that affects the sensor measurements. Accordingly, it is for sure not possible to apply a complete obstacle recognition logic if the vehicle is subjected to constant oscillations. Indeed, with an error threshold of 15 cm, small obstacles such as road bumps, ditches and steps could not be recognized; instead, obstacles that greatly increase or decrease the sensor measurements, such as slopes or large obstacles, would be easily recognized. The most dangerous obstacles, therefore, should still be detected by the sensor despite the numerous oscillations.



Graph 5.2.2: Measurements on dirt road

5.2.3 Pavé

Pavé is a traditional type of outdoor road paving made up of stone or porphyry cubes. The size of the pavé cubes is in the order of ten centimeters, therefore, even if it is a not very homogeneous pavement, it should not cause significant oscillations to the vehicle, since the dimensions of the bicycle wheels are far greater than the distance of the stones.
If we had used a scooter, we would most probably have had more wobble due to the small size of its wheels. Here too, I performed several tests to check the results of the measurements. The data in graph 5.2.3 are not very different from those obtained for the asphalt road as free road measurements are contained into the threshold levels.
From now on, therefore, for convenience, the pavé-clad inner courtyard area was used to perform the next obstacle tests.



Graph 5.2.3: Measurements on pavé

5.3 Phase II: obstacle recognition

In this second test phase, the code loaded on the device was modified by adding the possibility of recognizing obstacles just with the variation of the measured distance (regardless of the type and size of obstacles). I performed a series of tests using different types of obstacles including two cardboard boxes of different heights (26 cm and 34 cm), a cemented polystyrene cylinder (30 cm) and two 10 cm roll-ups.

Photo 5.3: Obstacles used in tests

As I didn't have a road bump available, two roll-ups were used next to each other to simulate the "bump case".
The graphs used show the measurements performed over elapsed time, the two threshold levels, the maximum values measured by the sensor and I also reported the behavior over time of the *"flag"* variable used to discriminate obstacles. In this phase only two values were used:

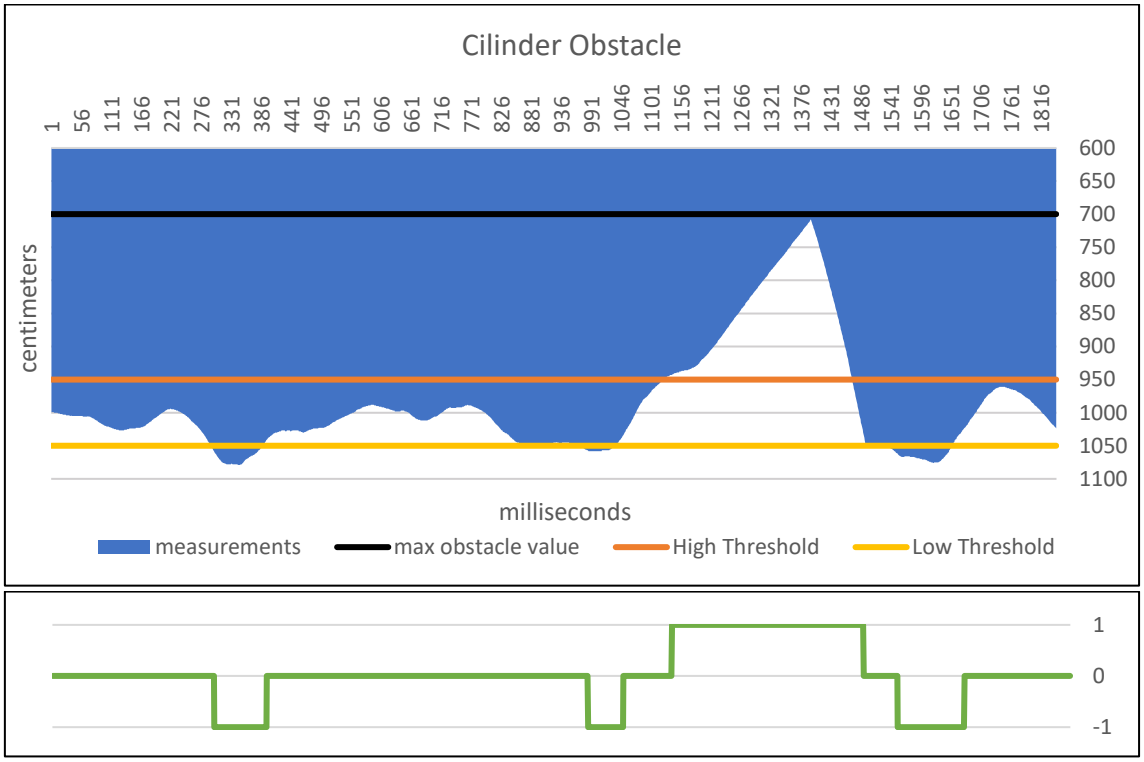
- **1** for the obstacles that exceed the high threshold.
- **-1** for the obstacles that exceed the low threshold.

5.3.1 Cylindrical obstacle

To correctly simulate the behavior of the sensor I used different obstacles made of different materials, sizes and shapes.
Initially, only one obstacle was placed in the middle of the road: the cylinder.
The data relating to the measurements carried out are shown in graph 5.3.1 below. The graph shows the presence of some disturbances during the ride that lead some measurements to exceed the lower threshold. This may be due to a simple "offset" caused by a slight displacement of the sensor. In fact, although the sensor has been rigidly fixed, it is always possible that it undergoes small displacements due to continuous percussion.
Despite the presence of these disturbances, the height of the obstacle was detected correctly since the minimum measurement detected by the sensor is 700 cm. Referring to the formula with which I computed the height of the obstacle, we obtain a result identical to the real height of the obstacle.

$$height = \frac{1000\ cm - 700\ cm}{10} = 30\ cm$$

Even if we consider an offset of 10-20 cm, the height of the obstacle would have an increase of 1-2 cm. Remembering that our goal is not to accurately detect the height of obstacles but only to detect the extent of the danger, the way to prevent an accident, we can say that the result obtained is still very good.



Graph 5.3.1: Measurements performed on cylindric obstacle

5.3.2 Set of obstacles

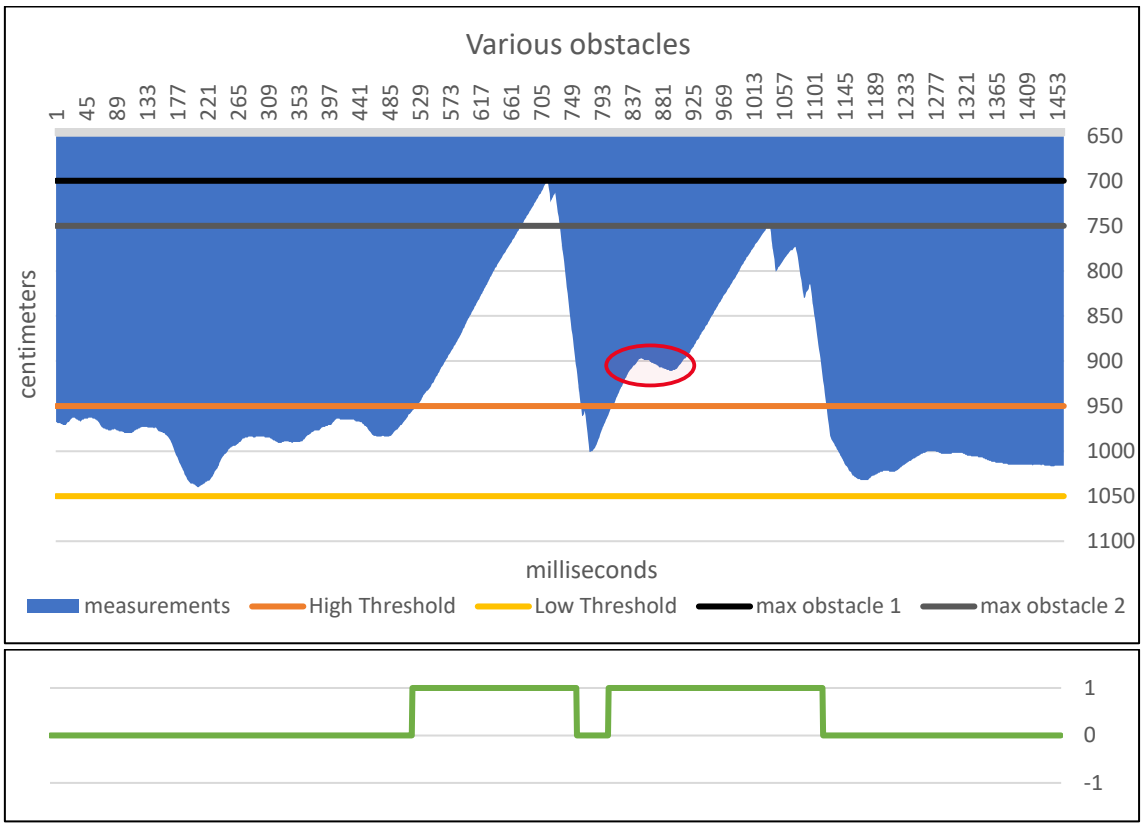
After testing using only one obstacle, I performed tests with multiple obstacles. In particular, in addition to the cylinder, I also used the 26 cm box and the 10 cm roll-up and I placed them as shown in the photo 5.3.2.

In this case there were no disturbances in the measurements and the sensor only detected real obstacles. The measurements were also very accurate because I detected maximum heights of 30 and 25 centimeters, as evidenced by the two straight lines (black and gray). The height of the second obstacle measured has an error of 1 centimeter compared to the real height of the obstacle, but it is completely irrelevant to our purposes.

$$height_{obs1} = \frac{1000\ cm - 700\ cm}{10} = 30\ cm$$

$$height_{obs2} = \frac{1000\ cm - 750\ cm}{10} = 25\ cm$$

Photo 5.3.2: Arrangement of obstacles



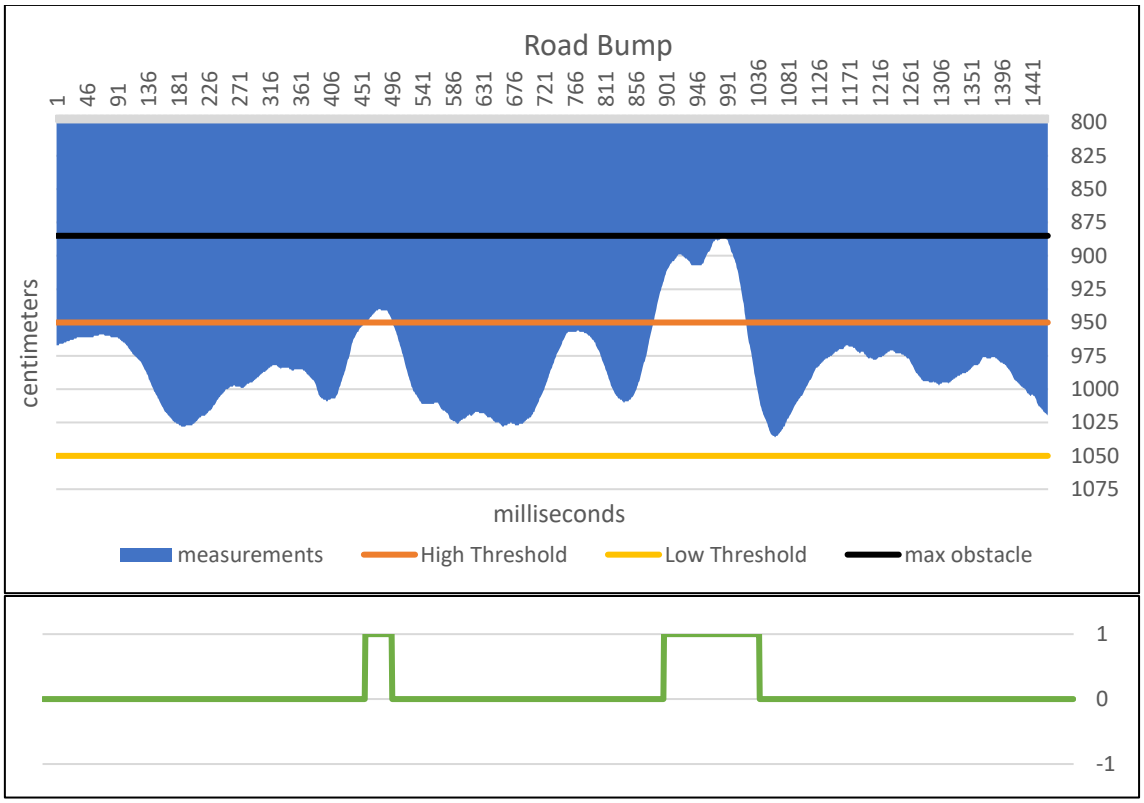
Graph 5.3.2: Measurements performed on a set of obstacles

In graph 5.3.2, above, it is also possible to note (red circle) how accurate the detection of the height of the roll-up was (1000 - 900 = 10 cm), although it is not recognized by the sensor as a distinct obstacle.

5.3.3 Road bump of 10 cm

To test the detection of a road bump, I used two 10-centimeter roll-ups placed side by side. It should be noted that road bumps with a height lower than 5 centimeters cannot be correctly detected by the sensor due to the set threshold levels. But, the real danger of road bumps arises when their size exceeds 10 centimeters: for example, in cities there are many raised pedestrian crossings that reach a height of 20 centimeters.

Graph 5.3.3 shows the result of the measurements made on the obstacle.



Graph 5.3.3: Measurements performed on a 10 cm obstacle

Looking at the graph, you can see how the sensor recognized the obstacle by recording a minimum measurement of 885 cm, that is, an obstacle height of 11.5 cm.

$$height_{\ bump} = \frac{1000\ cm\ -\ 885\ cm}{10} = 11{,}5\ cm$$

We could think of an error of a few centimeters but, later, I realized that the effective height of the roll-ups has been increased due to the presence of the handles of the case. So, the measurements are more than congruent.

Moreover, before detecting the obstacle, there was probably the presence of a slight oscillation which registered the presence of a false obstacle. Registering a false obstacle (a smaller and shorter one) is not a very serious thing, if we are talking about a few centimeters; the important thing is not to detect many of them, as the continuous signaling could disturb the driver.

5.4 Phase III: steps recognition

In the third stage, the sensor logic has been made more complex, adding the ability to recognize steps and sidewalks. Obviously, it will be possible to detect only differences in height greater than 5 centimeters since the sensor threshold levels are always valid. The graphs used show the value over time of the *"flag"* variable which, this time, can assume two more values than in the previous phase:

- **2** for up steps.
- **-2** for down steps.

In the area at my disposal there was a sidewalk that flanked both the pavé and the dirt road. I used, therefore, the sidewalk to perform the various tests. The sidewalk available was 15 centimeters high, as shown in photo 5.4 below.

Photo 5.4: Sidewalk height

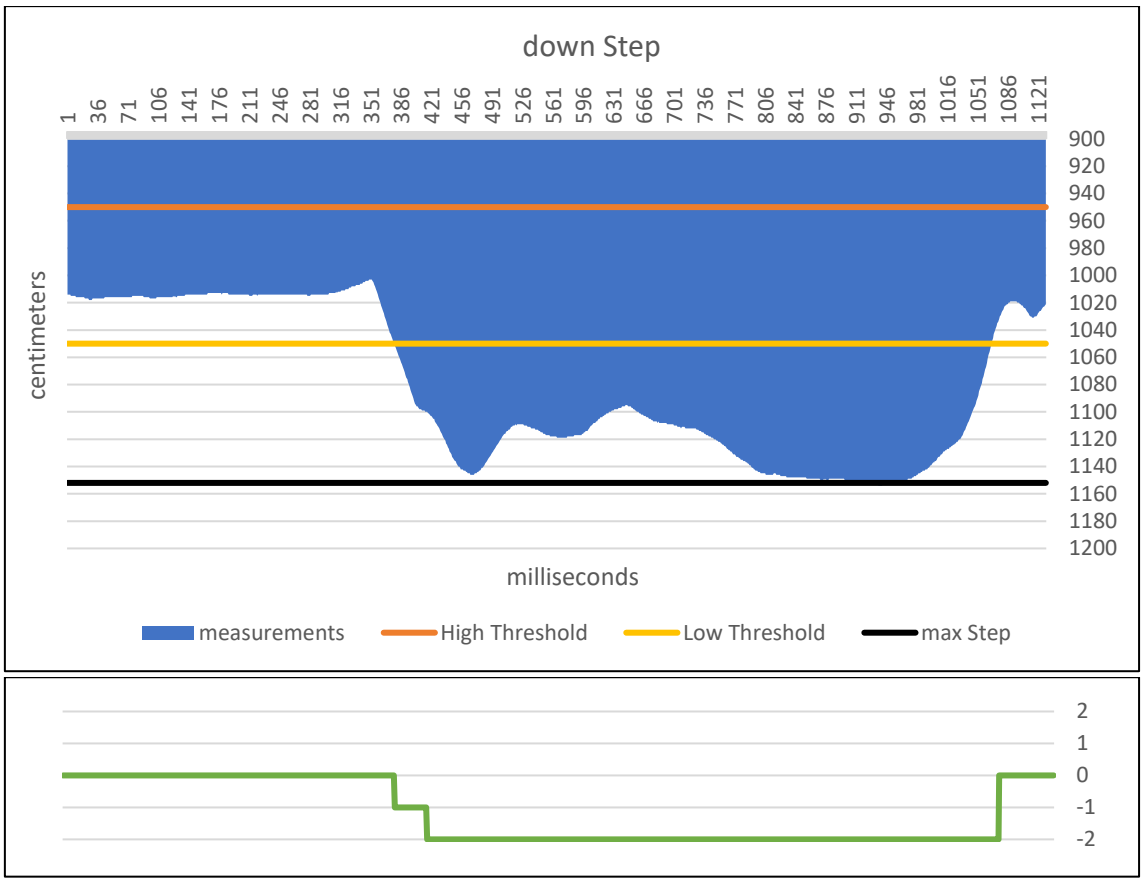
5.4.1 Sidewalk

Using the sidewalk, I performed a series of tests by detecting the measurements performed in the case of a down-step or an up-step.
Graph 5.4.1a shows the measurements obtained in the down-step case. The graph shows a peak of measurements of 1150 centimeters. By doing the calculations it turns out that the height of the detected step corresponds exactly to the height of the sidewalk.

$$height_{\ step} = \frac{1150\ cm\ -\ 1000\ cm}{10} = 15\ cm$$

The measurements made present some disturbances but remain constantly above the threshold, therefore, the *"flag"* remains set to the value *'2'* as it should be. The disturbances in the graph are due to the presence of dirt road after the sidewalk and

it is completely normal for similar oscillations to occur due to the differences in level of the ground. At the end, we can see the values return above the threshold since the road ends and there is a wall that fences the area.

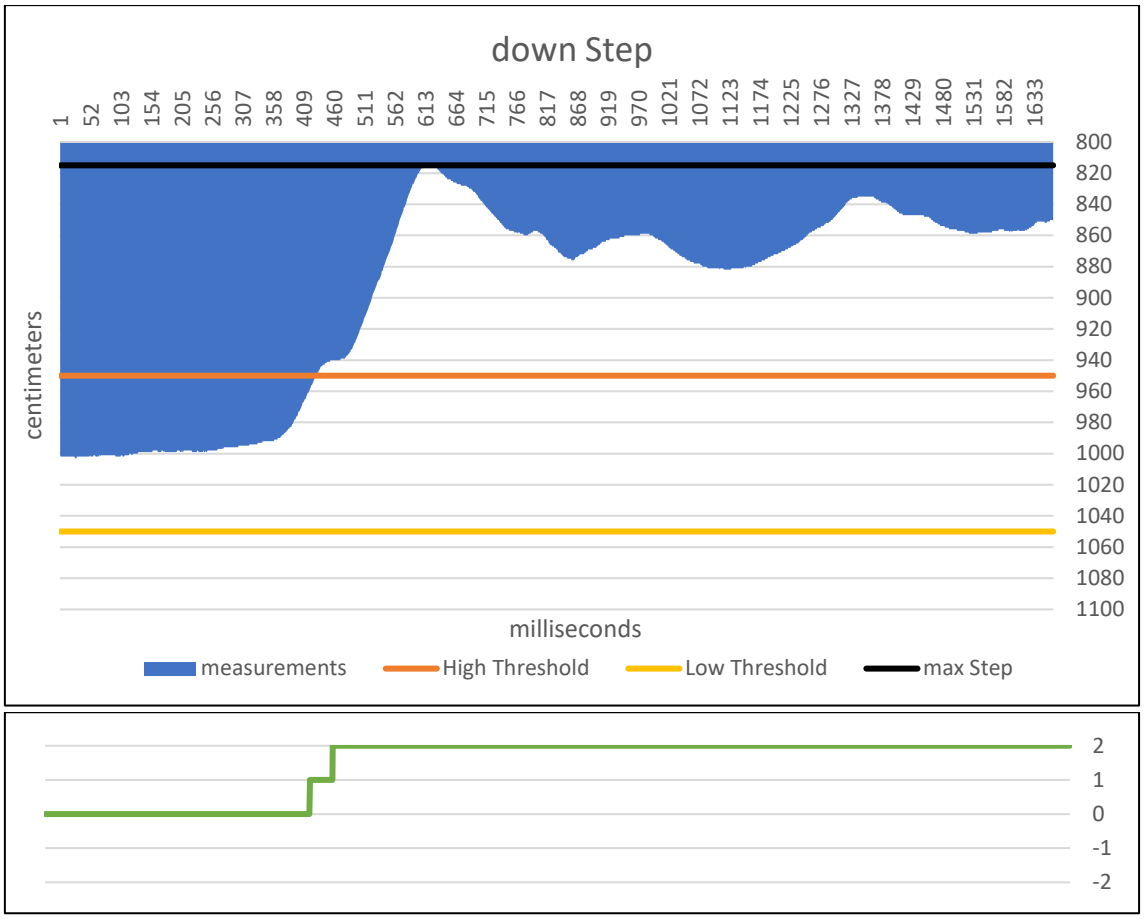


Graph 5.4.1a: Measurements performed on a down step

In the event that the vehicle was on the road and there was a sidewalk in front of it, the behavior would be equal and opposite. This time I performed the tests on the pavé side, for greater comfort in the trajectory. Graph 5.4.1b shows the data relating to the up-step case.
The measurements reported are in line with what we expected, even though I detected a peak 3 centimeters higher than the real size of the sidewalk. This time, therefore, the user will be alerted with the presence of a slightly higher bump than its real size.

$$height_{\ step} = \frac{1000\ cm\ -\ 820\ cm}{10} = 18\ cm$$

The cause of this may be due to a particular oscillation that occurred just as the laser beam was starting to detect the step. The measurements after this peak remain constant, even more than in the previous case and this is completely normal, if we consider that the vehicle is on the pavé and the current oscillations mirror those provided by the data in the free road graph 5.2.3.



Graph 5.4.1b: Measurements performed on an up-step

5.5 Phase IV: sensor complete logic

The last test phase of the project focuses on the detection of any type of obstacle, considering both stationary obstacles on the road and moving obstacles that can suddenly appear in front of the driver.
This is the reason why, now, I loaded the complete code of the sensor logic and in the graphs, thus, the *"flag"* variable can assume any value among those shown in paragraph 4.5.1.
To complete the logic, 3 more values were added:

- **3** for generic stationary obstacles.
- **4** for moving obstacles.
- **-3** for steep descents or escarpments.

Since it was not possible to simulate the steep descent case, the tests were performed only on the recognition of stationary or moving obstacles.

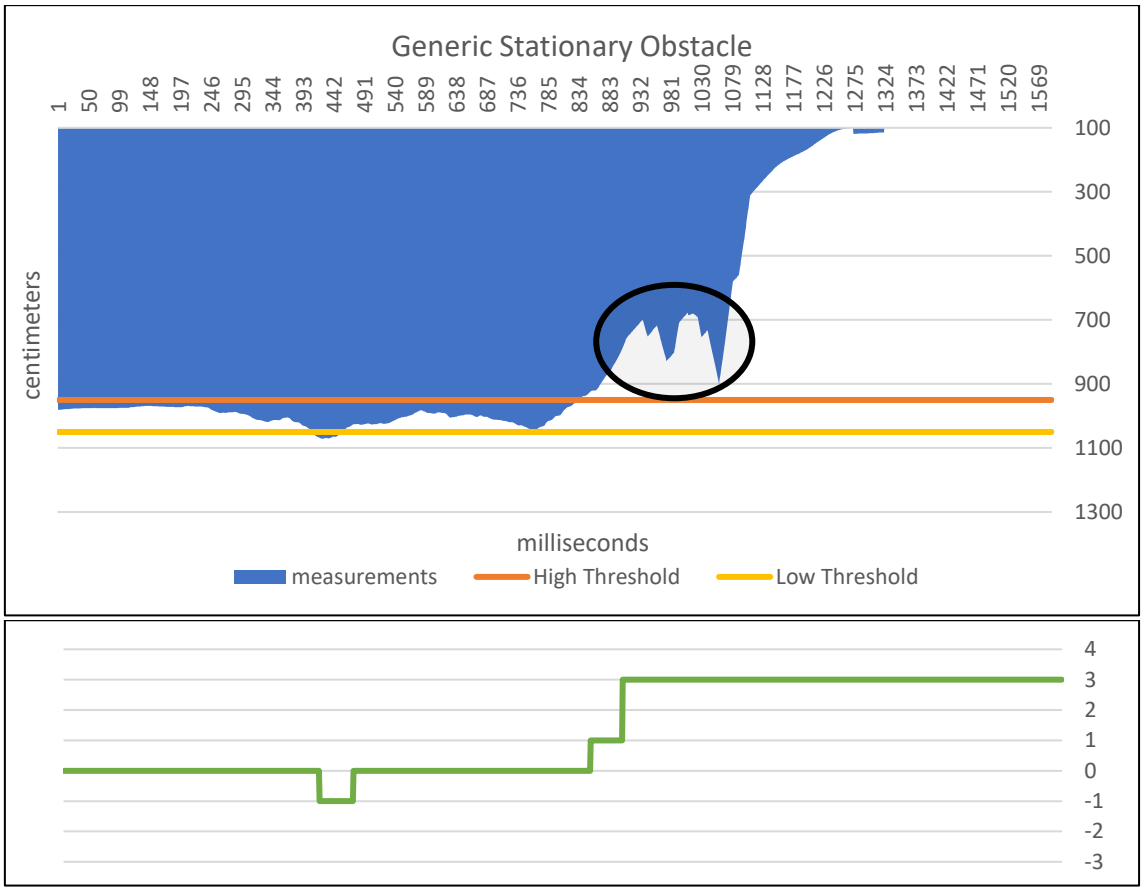
5.5.1 Generic stationary obstacle

To simulate the recognition of a generic stationary obstacle, I used the 26 cm cardboard box but it was placed standing on its longer side, in order to reach a height higher than that of the sensor. The precise height of the obstacle does not matter, because in this case the sensor will not be able to calculate it.

Photo 5.5.1: Obstacle arrangement

The generic stationary obstacle has to be detected by the sensor as an extension of the road bump case. Therefore, we expected the sensor to initially detect a road bump and, immediately after, to recognize the presence of a large obstacle blocking our trajectory. In Graph 5.5.1, the data shows exactly this behavior.
In the example shown in the graph, I made a front run against the obstacle and, for this reason, the sensor continued to detect the obstacle until I reached one meter from it.

I have chosen this dataset, in particular, to make it immediately evident what can happen while an obstacle is being detected. If we look at the part of the graph circled in black, we notice a sudden disturbance while the sensor is detecting the obstacle. This disturbance is due to the vehicle driving dynamics; indeed, the obstacle is 26 centimeters wide and it is enough to move the bicycle handlebar a few degrees to no longer detect the obstacle and have that strange disturbance. It cannot be called a disturbance because the measurements have not been altered but it is the driver who has changed direction and no longer points at the obstacle.
This is one of the problems that can occur when driving a very dynamic vehicle such as a bicycle. However, it must be taken into account that if a small swerve is enough to no longer detect the obstacle, it means that it takes very little to no longer have the obstacle in front of our trajectory. The bicycle, in fact, or even the scooter, are very slim vehicles: the bicycle wheels can reach a maximum of 10 centimeters while the handlebar occupies about 60 centimeters.
Despite this, the *"flag"* variable has always continued to detect the presence of an obstacle in that time frame because the average value of the measurements has not fallen below the threshold level, due to the effect of the *"distAvg"* average value.
Again, I detected a slight oscillation which led some measurements to exceed the low threshold level, signaling the presence of a false obstacle.

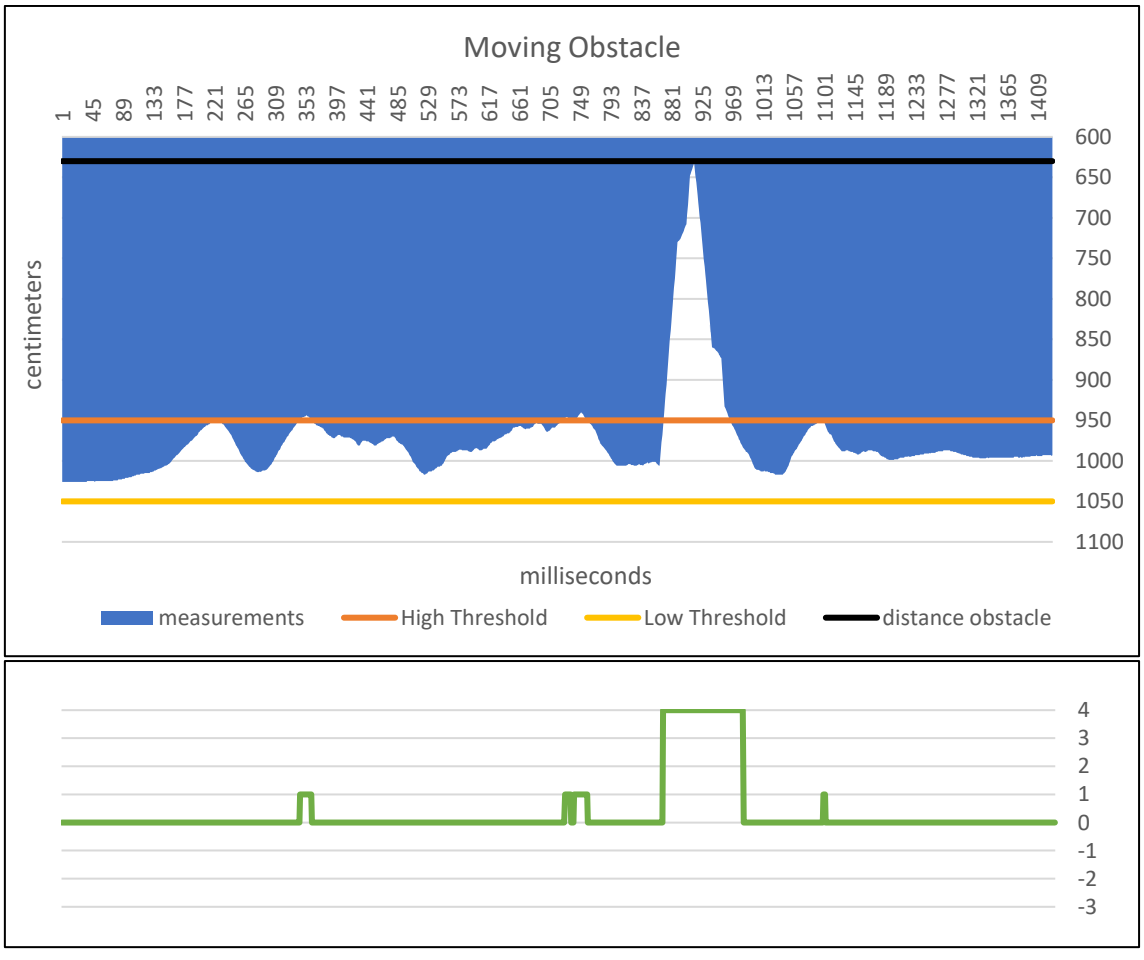


Graph 5.5.1: Measurements performed on a big stationary obstacle

It should be specified, as already mentioned in paragraph 4.5.7, that the obstacle could be either a stationary person or a wall, therefore, it is not possible to distinguish the different possible cases using only a single beam distance sensor.

5.5.2 Moving obstacle

To perform tests on a moving obstacle, I asked a colleague for help. While I drove the vehicle, he walked perpendicularly to my direction, cutting my way.
Graph 5.5.2 shows the case in which the obstacle suddenly passes in front of the vehicle without stopping. From the data used in the graph, you can notice a sudden surge in measurement values that allows the sensor to recognize the obstacle and signal it as a moving obstacle. In this case, the peak detected does not correspond to the true distance in which the obstacle crossed the vehicle trajectory, since I used the average distance in the graph. But, on the contrary, I exploited the real distance value for the acoustic and visual signaling, thus, allowing to provide the user with the right warning of danger based on the obstacle distance.



Graph 5.5.2: Measurements performed on a moving obstacle

In the event that the obstacle remains in the path of the vehicle, the sensor would, however, continue to signal to the user the proximity of the obstacle as it comes closer to it, exactly as in the case of a generic stationary obstacle, using an acoustic signal acoustic with higher frequency according to the danger. Still here, there are some slight oscillations that lead to the detection of small false obstacles.
Since all the disturbances obtained in the graphs are caused by oscillations that bring the measurements to exceed the threshold level by no more than 10 centimeters, I came to the conclusion that setting the two threshold levels to 60 centimeters (instead of 50) could solve the problem. However, it must be taken into account that the range of detected obstacles is further reduced, making undetectable all obstacles with a height lower than 6 centimeters.

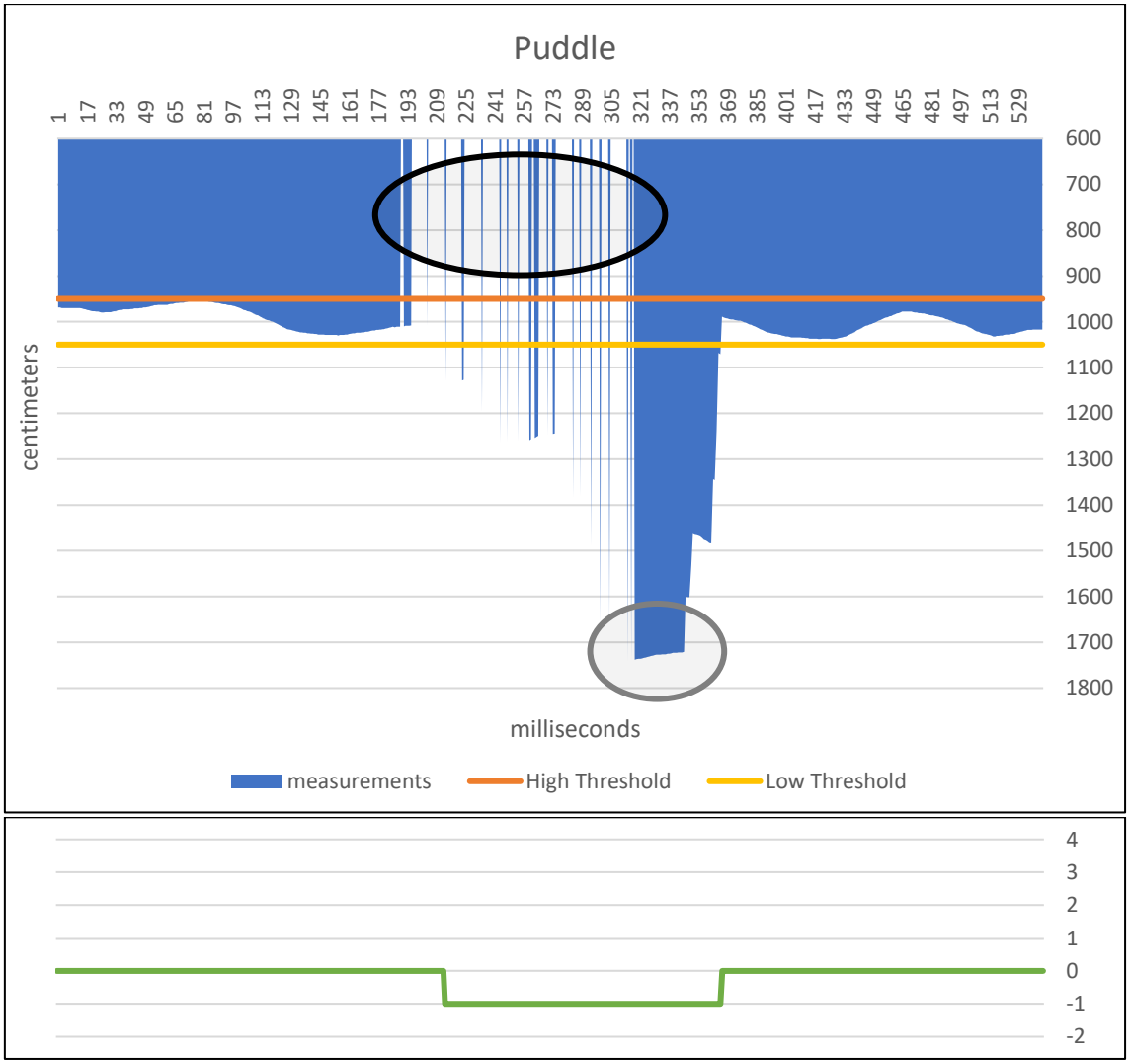
5.5.3 Climatic conditions influence

The testing phase of the sensor prototype was carried out in the last months of my thesis project (from mid-February until the first days of March). Unfortunately, in this time period, there were no adverse weather conditions: there were no rainy days, except very light rainfall (almost imperceptible). Before the test phase, the device was briefly tested during days with heavy rain and even snow. Unfortunately, the sensor was not yet equipped with the SD card module and it was not possible to store the measurements and errors detected by the sensor. Therefore, I could not analyze the data and create graphs, but I can say that, on those days, looking at the error messages on the LCD display, huge amounts of errors were detected. The sensor was unable to perform the measurements correctly, both in the rain and, especially, in the snow.

“Why this behavior?”

The distance measured by the sensor depends on the reflected energy that is detected by the receiver and, in case of precipitation, the reflected laser beam may be disturbed and the receiver may not receive the signal correctly. In addition, the presence of liquids on the road could cause inadequate dispersion of the laser beam. For this reason, liquids can be a relevant problem in obstacle detection.
In case of a flat and highly reflective liquid surface, the reflected energy of the laser may not be detected unless the beam hits that surface from normal. On the contrary, small surface ripples can create an enough dispersion of the reflected energy to allow the detection of the liquid even if the laser beam comes from other directions. [15]
For this reason, I decided to carry out tests by spraying water on a pavé stretch simulating the behavior of the sensor, the way to check whether the presence of liquids on the road could affect the measurements performed.
Graph 5.5.3 shows the data obtained and a conspicuous presence of errors was recorded during the piece of wet road. The white area in the graph (circled in black) highlights the absence of measurements performed correctly. Furthermore, the number of measurements taken has not been only decreased significantly, but those

measurements have also been altered due to the reflective effect of the water on the road.
This problem should not be underestimated, in fact, as you can see in the graph, a measurement peak of more than 1700 cm is reached; this signaled the presence of a 70 centimeters false obstacle (circled in grey).



Graph 5.5.3: Measurements performed on wet pavé

The results obtained show us how difficult it is to obtain correct measurements when the road is wet. For “wet road” I means a surface sprinkled with water and not simply humid.
In case of heavy rainfall, probably, there would be errors not only due to the presence of water on the road, but also due to water drops or snowflakes that cross the laser beam and significantly alter the probability of performing a measurement correctly.

5.5.4 Obstacle reflection problems

Obstacles of different materials, shapes and sizes were used throughout the test phase and no obstacle reflection problems were detected. From the data obtained from tests performed, no measurement error was detected; this means that the signal strength has never been compromised and the sensor has always correctly performed the measurements of obstacles on all types of road surfaces under examination.

The only measurement errors were found indoors, in a very large room with a perfectly flat and smooth floor, when I was taking measurements to verify the correct functioning of the sensor. The causes of these errors have not been analyzed in detail since the sensor was designed to operate on vehicles, and therefore outdoors. Probably, it would have been very interesting to carry out tests on a road covered with smooth marble blocks, typical of some historic centers.

Conclusions

During the test phase, the sensor proved to be very reliable for the detection of obstacles on asphalted or paved roads (such pavé).
The sensor was able to recognize the dangers by also detecting in most cases the exact height or depth of the obstacles. Unfortunately, in order to avoid the detection of false obstacles, it was necessary to set a threshold of 60 cm which does not allow to detect obstacles with a height of less than 6 centimeters. This is due to the numerous oscillations that the vehicle is subjected to while driving.
The sensor prototype was built entirely with my hands and however rigid the iron structure created may be, it is subject to small inclinations which affect the distance measured by the sensor by means of an offset. In addition, the shock absorbers on the vehicle can create an undulatory movement of the entire vehicle, consequently changing the height and angle of inclination of the sensor which, therefore, no longer constantly detects the 10 meters distance.
Probably, the use of a more rigid vehicle such as a traditional bicycle or an electric scooter could reduce the amount of oscillations due to the effect of the shock absorbers, but also increase them due to the absence of the same. Tests should be carried out on each vehicle and an appropriate threshold level should be chosen for each one.
For sure, the sensor cannot be used adequately on unpaved or uneven roads such as dirt roads, since the large number of oscillations involves such a high error as not to allow correct recognition of obstacles. The latter does not represent a measurement error, as the sensor has always detected the distances correctly during the tests, but it is an error due to the sensor position which, moving, compromises all the calculations performed to create the logic of obstacle detection.
The sensor, on the other hand, was designed to be used in urban centers where there are mostly asphalted and paved roads.
By running the tests at a speed of 10 km/h, the sensor was able to correctly detect obstacles. It was not possible to carry out tests at higher speeds as there was not too much space available, but since the number of measurements per meter performed inversely depends on the speed of the vehicle, doubling the speed you halves the number of measurements. To keep the sensor efficiency constant and correctly detect obstacles even at high speeds, a more powerful microcontroller should be used, capable of minimizing code execution times.
The presence of adverse climatic conditions can considerably compromise the operation of the sensor both by creating voids of measurements (the sensor cannot perform the measurements correctly) and by altering their value (the sensor detects false obstacles). On the other hand, with the absence of precipitation, the sensor has always performed correctly the measurements on any road and with obstacles of different shapes and sizes.
In summary, if a moderate speed is maintained, the designed sensor will be able to correctly detect obstacles and provide the user with information and alerts in real time.

At the end of my thesis project, given the results obtained in the testing phase, I was able to ascertain the feasibility of the POC.

References

[1] Monopattini elettrici, con la diffusione aumentano gli incidenti, La Stampa, 12 Agosto 2020 di Alessandro Vai, https://www.lastampa.it/motori/attualita/2020/08/12/news/monopattini-elettrici-con-la-diffusione-aumentano-gli-incidenti-1.39186873

[2] Bonus Mobilità, Ministero dell'Ambiente, 2 Novembre 2020, https://www.minambiente.it/comunicati/bonus-mobilita-dalle-9-del-3-novembre-si-potra-accedere-al-portale

[3] I 10 pericoli più frequenti per chi va in bici, 28 Aprile 2020 di Suva, https://www.suva.ch/it-ch/news/infortunio/i-10-pericoli-piu-frequenti-per-chi-va-in-bici

[4] Le regole per i monopattini elettrici, Ministero dell'interno, 16 Marzo 2020, https://www.interno.gov.it/it/notizie/regole-i-monopattini-elettrici

[5] Spazio di Frenata, Youmath, https://www.youmath.it/domande-a-risposte/view/6684-spazio-di-frenatura.html

[6] Come funzionano i sensori ad ultrasuoni, Beijing Ultrasonic, 18 Aprile 2017 di Jessie Wong, https://www.bjultrasonic.com/it/how-do-ultrasonic-sensors-work/

[7] Conoscenza dei sensori a ultrasuoni, Pepperl+Fuchs, 21 Gennaio 2014, https://www.pepperl-fuchs.com/italy/it/24907.htm

[8] Shawn, Types of Distance Sensor and How to select one, Seeedstudio, https://www.seeedstudio.com/blog/2019/12/23/distance-sensors-types-and-selection-guide/

[9] IR Sensor Working and Applications, Robu.in, 19 Maggio 2020 di Priyanka Dixit, https://robu.in/ir-sensor-working/

[10] Tecnologia LiDAR: che cosa è? come funziona?, Consystem, https://consystem.it/faq/tecnologia-lidar-che-cosa-e-come-funziona/

[11] LIDAR: cos'è, come funziona e a cosa serve la tecnologia, Internet4things, 19 Maggio 2020 di Eugenio Tommasi, https://www.internet4things.it/iot-library/lidar-cose-come-funziona-e-a-cosa-serve/

[12] How LIDAR Based ADAS Works for Autonomous Vehicles, Einfochips, 27 Dicembre 2018 di Anshul Saxena, https://www.einfochips.com/blog/how-lidar-based-adas-work-for-autonomous-vehicles/

[13] LIDAR-Lite v3, Garmin, https://buy.garmin.com/it-IT/IT/p/557294/pn/010-01722-00

[14] LIDAR-Lite v3, Sparkfun, LIDAR-Lite v3 - SEN-14032 - SparkFun Electronics

[15] LIDAR-Lite v3 Datasheet, Garmin, LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications.pdf (garmin.com)

[16] Elegoo UNO R3 Datasheet, ELEGOO, Datasheet\ELEGOO UNO R3 Board.pdf

[17] LCD Display, ELEGOO, Datasheet\2.13 LCD Display.pdf

[18] RGB LED, ELEGOO, Datasheet\2.2_RGB LED.pdf

[19] Passive Buzzer, ELEGOO, Datasheet\2.6_Passive Buzzer.pdf

[20] Micro SD card Module, Adrirobot, 02/01/2019, Micro SD Card Module (adrirobot.it)

[21] Powerbank, Mbuynow, https://www.amazon.co.uk/20000mah-Mbuynow-Laptop-Battery-Samsung/dp/B07D6LP7T1

[22] Arduino board, Wikipedia, 07/03/2021, https://it.wikipedia.org/wiki/Arduino_(hardware)#Arduino_IDE

[23] Arduino IDE, Wikipedia, 21/11/2020 https://it.wikipedia.org/wiki/Arduino_IDE