A Multi-Sensor Approach to Optimise Mountain Bike Suspension

Jérémy Lesauvage

Master's thesis in computer science

Åbo Akademi University
Faculty of Science and Engineering
Information Technologies

June 2020

Supervisor: Sébastien Lafond

Abstract

The Mountain bike sport is an off-road sport, with distinctive aspects, from the family-friendly excursion on the forest roads to the disciplines recognised as extreme, combining obstacles such as jumps, roots, rock gardens and other rough terrains. The suspension is a primordial piece of equipment on the mountain bike to provide dampening, comfort, traction, and efficiency over those cited obstacles. They are designed to be versatile through different adjustments to accustom to any situation, trail, rider's weight, and preferences.

Even though the process of setting the suspension itself is effortless, it is the users' responsibility to optimise it, based only on their feelings, since no affordable tools or universal solutions are provided to them. This thesis introduces the design and implementation of a multi-sensor approach to back up or contrasts the user feelings when tuning the suspension.

This design proposes a two-part solution. First, the on-bike electronic, composed of distance sensors to keep track of the fork travel over times, working with an Arduino Mega board, transmitting data through a Bluetooth module to an Android app to save the data. The sensors are paired with a GoPro Hero 7 to provide synchronised video footage as well as another acquisition method. The second part is the computer environment where the sensor's data will be exported from the Android app and analysed through a Python script, as well as the data of the video footage extracted via a motion analysis software.

One main goal of the thesis is to conclude if we could uniquely adopt a GoPro to optimise the suspension, leading to a competitive price for such a product and even free since most riders already own a GoPro. For this purpose, a comparison with the sensors' results will be provided and discussed.

The expected results will be easy to understand written suggestions, to lead to an optimised setting, and graphics displaying the following data acquired:

- Shaft compression (mm) / time (ms)
- Shaft velocity (m/s) / time (ms)

From these graphics, we can analyse the four different damping adjustments: low-speed/high-speed compression and rebound, but also if the user's suspension exploits all its travel, the bottom-out resistance, and its capability to return to SAG value.

Keywords:

Mountain bike telemetry, Suspension tuning, GoPro, Accelerometer, Distance sensors

Table of content

Figures

Tables

Acknowledgement

I wish to express my deep gratitude to my master thesis supervisor, Mr Sebastian Lafond researcher & lecturer at Åbo Akademi, for his confidence in my atypical choice of thesis subject.

I would like to show my appreciation for the assistance he offers me through each stage of the process by giving me valuable advice and feedback. In addition to this, I wish to thank him for his availability through our weekly online meeting, not only he was helping me with my thesis writing but also with academical related topics with patience, calm and kindness making my way through my master's degree easier and reassuring.

I wish to show my gratitude to my friend Jean-Baptiste Verdret for his continuous support during my academic years, but more particularly for his recent review of my Python code helping me to achieve higher standards.

Finally, I am indebted to my parents who infallibly supported me through my academic choices and my passion for mountain biking by financially providing me with the fork used in this thesis.

List of abbreviations

HSC	High-Speed Compression
HSR	High-Speed Rebound
LSC	Low-Speed Compression
LSR	Low-Speed Rebound
MTB	Mountain bike
PSI	Pound-force per square inch
IN	Inches
LBS	Pound
HMI	Human Machine Interface
DJ	Dirt Jump
XC	Cross Country
TR	Trail
EN	Enduro
DH	Downhill
ToF	Time of Flight
T1SD	Test 1 Small Drop
T2SH	Test 2 Successive Hits
T3TC	Test 3 Technical Climb
T4J	Test 4 Jump
TT1	Test Trail 1
VCSEL	Vertical Cavity Surface Emitting Laser
ETRTO	European Tyre and Rim Technical Organisation
IES	Electronic Image Stabilisation

1. Introduction

The following introduction contains a presentation of the mountain bike disciplines and their specifications in terms of suspension, which leads us to talk about low and high-end forks and their diversity of adjustability. This presentation is complete, we can then describe the problem to solve, as well as the scope and limit of this thesis project.

1.1 Presentation of the problem

Suspensions are the vital equipment of a mountain bike (MTB). Rightly, they offer the riders comfort and the piloting precision they need to tackle obstacles such as jumps, roots, rock gardens and other rough terrains and trails, or even grab the maximum traction and pedal efficiency.

The market of MTB front suspension is divided between about a dozen companies, such as DT Swiss, Öhlins, DVO and Marzocchi, even though, the two market leaders are incontestably Fox Suspension and Sram RockShox [1] as shown in Figure 1. The reason behind the popularity of those two brands is the excellent coverage of each mountain bike discipline through their products.

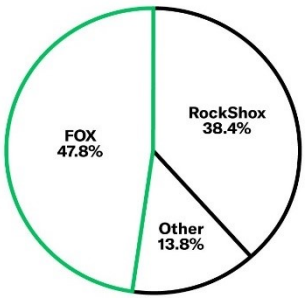


Figure 1: Fork sale by brand 2019, © TheProsCloset

Indeed, each company designs their forks for a specific MTB discipline: Dirt Jump (DJ), Cross Country (XC), Trail (TR), Enduro (EN), Downhill (DH). The most crucial difference is the number of travel, respectively, around, 80, 100, 130, 150/160, 200mm, resulting in fork height difference. More travel means a more capable fork that will have more resources to dampen the terrain, but in counterpart, they are heavier and less pedal efficient. To resume, the fork choice will dictate the discipline the bike is designed for, and therefore the type of settings, indeed, cross country riders will seek pedal efficiency, and speed conservation, whereas downhill riders will aim for comfort and obstacle clearance.

In the past, coil spring forks were popular due to easy conception and maintenance, but in recent years, most companies are working on air spring forks. Air spring forks possess disadvantages, such as not being as plush and maintenance-free as the coil forks due to frictions with more seals.
However, companies judged that those drawbacks were outweighed by the advantages air spring forks offer, such as a lighter fork: up to 300g, and a more adjustable fork. For comparison, the recent average enduro fork weight is around 2,1kg, which in this case will represent a weight saving of 15%. In an air spring fork, the user can fine-tune the amount of compressed air inside the air chamber. In contrast, the coil spring forks only propose preload adjustment, to change the spring rate of the coil fork the user needs to change the spring physically.

Spring rate: the force needed to compress a spring for a certain distance, in metric (kg/mm) in standard notation (lbs/in). The higher the value is, the stiffer the spring is.
Preload: Act of compressing the spring without affecting the fork travel, more preload means a more compressed spring resulting in a firmer suspension.

In the past few years, coil spring forks were only available as cheap solutions for mountain bikers, but Vorsprung and Push Industries have proposed a coil spring conversion kit providing an option for the rider to transform an air fork to a coil fork promptly. The riders were subjugated by how much comfort and traction they could obtain, thus beginning the comeback of coil-spring forks. For enduro and downhill, the most extreme discipline, sacrificing weight over comfort is a compromise that some companies like Marzocchi are willing to take by proposing a high-end fork available in air or coil spring such as the Marzocchi Bomber Z1. Another alternative spring technology is the dual air spring, which enables the user to adjust the positive spring and the negative spring pressure separately, while in a solo air spring the negative spring is equilibrated to the same pressure as the positive air chamber. The solo air spring is the simplest solution, which resulted in the dual air spring being abandoned, except for Cane Creek who propose the Helm, a high-end dual air spring fork model trying to exploit this extra adjustment.

Even if the coil-spring forks recently came back, they still represent a low percentage of sold forks, and therefore in this thesis, we will focus on the most popular fork technology and brand, the Fox suspension equipped with a solo air spring.

Coupled with the importance of spring side technology, forks designers work on the damper side too, to achieve peak performance or simplest solution to create different options to help reduce prices to reach a broader range of customers. Consequently, low-end forks offer fewer adjustments, usually spring rate, and rebound without distinction of high-speed nor low-speed. On the other hand, high-end front suspensions have up to six settings; such a fork can be tuned through external knobs that have several “clicks” representing a small amount of rotation that will open or close an orifice to control oil flow. The Fox Grip2 cartridge is one example, see Figure 3, 4.

Following is an explanation of what the different settings enumerated above are and how they act on the bike characteristics.

Air spring pressure increases the firmness or plushness of the spring to adjust the suspension to set up the SAG. SAG is the percentage of distance the suspension compresses under the weight of the rider in a neutral and static position on the bike. SAG is the most critical setting and must be done first, with all the other settings fully open (0 clicks). SAG must be checked regularly as the rider weight is inconstant and seals are defective over time.
Too little SAG results in a firm suspension that may lead to rider exhaustion, whereas too much of it creates a risk of bottom out and no support in corners. The recommended SAG depends on the discipline.

XC	TR	EN	DH
10-18%	20-28%	25-35%	30-40%

Table 1: SAG Values

Figure 2: Settings position, © Fox Racing Shox

Compression controls the forces that actuate the fork, and compression settings are illustrated in Figure 3.

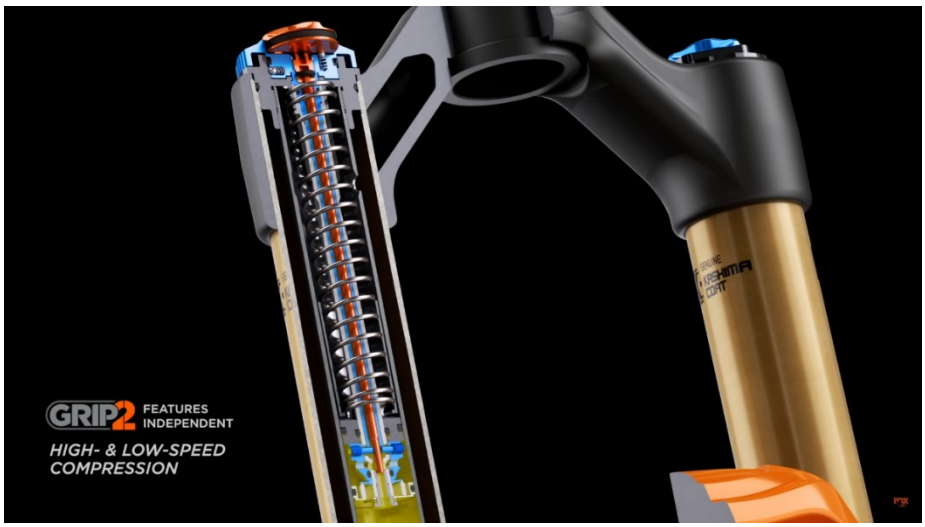


Figure 3: Grip 2 LSC-HSC adjustment, © Fox Racing Shox

Low-speed compression (LSC) happens when the fork is slowly compressed. LSC is to dissociate with bike speed. LSC manages small bumps, berms, corners, as well as pedal bob. Pedal bob occurs when the user stands up to pedal, a portion of the user's leg power is wasted in suspension activation instead of entirely going into the transmission. In fact, pedal bob is lost energy; and needs to be minimised as much as possible. LSC correctly adjusted can create a supple suspension; otherwise, LSC can produce pedal bob, tyre skidding and chatter, the feeling of never having traction in long areas of continual bumps.

High-speed compression (HSC) manages abrupt impact resulting in a quick compression of the fork, such as jump landing, but also including rocks, roots, and other terrain input. HSC prevents suspension from bottoming out. HSC correctly adjusted can lead to a plush suspension, absorbing irregularities of rough terrain with ease. Too low HSC leads to bottom out, resulting in a control loose and even in the rider being ejected off the bike, whereas too high HSC generates a too-firm suspension, resulting in the fork's travel being unused.

Rebound regulates the bounce-back speed; rebound settings are illustrated in Figure 4.

Figure 4: Grip 2 LSR-HSR adjustment, © Fox Racing Shox

Low-speed rebound (LSR) goal is always to return the fork to its SAG, too few LSR and the fork keeps compressing under multiple stroke/impact, too many and the fork is too bumpy making the tyres leave the ground and resulting in traction loss.
High-speed rebound (HSR) manages the fork extension on the backside of rocks, roots, brake bumps to help the tyre stick to the ground, thus providing traction in all situations.

Volume spacers are plastic pieces placed inside the air chamber of the spring to decrease the volume of air, thus increasing the mid-stroke and bottom-out resistance without interfering with the first part of the fork travel.

To demonstrate the number of possible combinations I refer to the most recent and high-end Enduro fork produced by Fox Suspension, the Fox Float 36 grip2 2020, offering:

- LSC: 14 clicks, HSC: 22 clicks
- LSR: 14 clicks, HSR: 8 clicks
- Air spring pressure up to $120psi$
- Up to 6 volume spacers [2]

Which represents $14 \times 22 \times 14 \times 8 \times 120 \times 6 = 24{,}837{,}120$ possible combinations (with increments of 1psi for the air spring pressure).

1.2 Motivation

A more considerable number of adjustments permits to obtain forks with a rich versatility satisfying indecisive person or a person with an atypical stature, tall, heavy, featherweight. However, it can be detrimental to the final user as few understand the theory behind the settings, leading to a lack of knowledge of how to adjust suspensions. Even if the suspension tuning itself is effortless (turning a knob, pumping air), the theory is required to understand each adjustment, and it is improbable to obtain the perfect optimised settings by luck, as shown with the number of possible combinations, and neither by feeling. Lack of tools to backup riders' intuition is one reason why riders are unable to optimise their suspension. Instead, they tend to rely on one average setting, tuned by feeling. The process of tuning the front suspension is not time-consuming and straightforward, requiring only a few tools (high-pressure pump and a wrench). If time or lack of tools are unaccountable for preventing the users from fine-tuning their suspensions, it might be due to lack of data, and no metric to determine which amount of adjustment is needed for a specific trail. I believe that if the users could visualise how their suspension is reacting to the trail they are riding, as well as the weather condition, rider weight and riding style, they would better understand the impact of their tuning. They could win in terms of comfort and performances and could achieve, with the help of suggestions, optimised settings.

1.3 Definition of the work

First, this work tends to help the user understand the behaviour of his suspension with the help of graphics, based on actual data acquired by the distance sensors and the GoPro. Real-size object determination will be applied to the GoPro footage on the fork stanchions to keep track of the fork travel over time. The second objective of this implementation is to analyse the acquired data for displaying tuning suggestions to lead to an optimal suspension setting. Finally, the last objective is to conclude on the possibility only to adopt a GoPro to optimise a front suspension, by comparing the result of the GoPro with the result of the sensors.

1.4 Scope and limits

This solution will ignore the rear suspension; the reason is that the sensor setup could change drastically for every frame since all bike manufacturers develop different frame designs and work with several suspension designs, while the front suspension is more standardised and possesses a sliding movement regardless of the frame. Furthermore, the sensors operated to determine fork travel are often unadaptable to the rear shock, as part of the frame, or shock mounts could interfere with the sensor as well as the pivoting movement of the suspension design induces difficulty to handle the time-of-flight sensor solution.

This solution is designed for the Fox grip2 single air spring fork; although the embedded system could be mounted on a dual air or coil-spring fork and benefits from the software, it is to the users' discretion to adapt the results to their setting (negative air chamber pressure, spring rate, spring preload). This solution is, however, incompatible with atypical forks such as the following.

Linkage fork (Structure Cycleworks, Trust, etc.):

The front suspension of the Structure Cycleworks design is based on a frame swingarm with cartridge bearings. A rear shock is used as the damper. Due to its concept, the fork offers 40% less brake dive (compression of the fork when the front brake is actuated) [3]. This solution prevents the usage of GoPro utilisation and therefore is unsuitable for our sensor implementation.

Figure 5: Linkage fork, © Structure Cycleworks

Lauf Grit fork:

The Lauf Grit is a light carbon fork design for Gravel and XC bikes. They are maintenance-free, the wheel axle is isolated from the main body by an S2 glass-fibre spring [4], with predetermined properties. There is no tuning involved since there is no adjustment.

Figure 6: Lauf Grit fork, ©Laufcycling

Inverted fork:

Figure 7: RS1 inverted fork, ©RockShox © mtbiking

The term inverted fork refers to a fork where the stanchions are placed between the wheel axle and the fork's main body, such as the RockShox RS1 or Cannondale Lefty.

The motorcycle fork inspires them. The advantage of this design is that the seal is continuously bathed in lubrication oil [5] by gravity, resulting in a better life span and greater sensitivity over small bumps. However, our sensor implementation is inappropriate for this type of fork, since no clear vision of the stanchion for the GoPro footage could be provided without an angle interfering with the calibration method of object size determination software.

2. State of The Art

The state-of-the-art chapter focuses on the theoretical background of the thesis work, which science, and engineering knowledge I am applying. Besides, it implies the study of similar products, company-oriented as well as individual works and lists the advantages and drawbacks of each product. I then review all the possibilities in terms of distance sensors suitable to my project.

2.1 Theorical Background

Telemetry systems are embedded systems combining hardware and software to provide access to data where it was impossible before. They are using a data science methodology: acquire, organise, visualise the data, and build a model. The hardware is composed of different sensors (temperature, pressure, distance, etc.), using communication protocol SPI and I2C, and an acquisition unit (Arduino, Raspberry, …).

The GoPro footage needs to be analysed to monitor the fork travel position and velocity. For this purpose, we can employ motion analysis software. Such software will track a moving point of a video and save its position in pixel frame per frame into a .csv file. To obtain value in mm a reference of a known dimension should be used. We can declare the reference with the constant distance of uncompressed fork stanchions. Some tests will be conducted to state which frames rate/stabilisation combination of the GoPro proposes the best results.

In this thesis, we focus on fork telemetry for casual riders, since this portion represents most riders. The main points of focus here are accessibility and reasonably priced solutions. Some companies thought the same and are targeting casual riders by providing an easy-to-use solution, but sometimes expensive. First, we will review those solutions focusing on the operating sensor.

2.2 Similar Telemetry System

Mountain bike telemetry appeared for world cup athletes, it helps their mechanicals to determine precisely how the suspension should be tuned for a specific race, but also, to monitor brake temperature and lean angle. Those data could be analysed to determine which pair of brake pads, rotors diameter, and tyre to utilise. Telemetry system results also play an important role in product development.

SuSS My Bike, SMB Flow Kit:
To monitor shock displacement, the "SMB flow" exploits a draw-wire displacement sensor solution. Such sensors are composed of a measuring cable, a spring-loaded spool, and a rotational sensor (potentiometer or encoder) [6], as shown in Figure 8 [7]. When a linear force actuates the cable, the spool rotates while the spring maintains tension on the measuring cable. With the rotation sensor and the constant diameter of the spool, we can determine the displacement of the cable and its velocity.

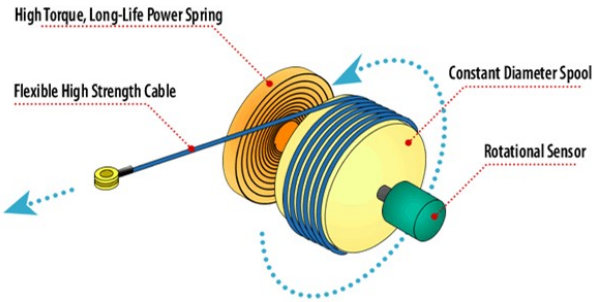


Figure 8: Draw *wire sensor components, © TE connectivity*

Figure 9: Suss My Bike telemetry solution, © SuSS My Bike

On the SuSS My Bike solution, Figure 9 [8], the sensor is attached to the suspension's main body and the cable-end on the other extremity. The SMB Flow Kit is designed to work with both fork and shock, regardless of air or coil spring due to software flexibility. Furthermore, this system works with inverted forks, and even on motorbike suspension. The embedded part works conjointly with a mobile application that authorises the user to define the available settings of his suspension, thus permitting higher compatibility and accurate tuning suggestion.

The other role of the mobile application is to transfer and analyse the data, as well as display suggested suspension settings and SAG settings. The only problem with this design is that the sensor kit can be difficult to mount on coil shocks as well as on some frame's designs using a shock tunnel, for example, the Santa Cruz Nomad 2021. However, only one device manages shock and fork tuning, since this waterproof draw-wire sensor permits to measure small displacement ideal for shock (between 38mm and 89mm), and more significant displacement adapted to fork (from 100mm to 203mm). The idea of visualising and tuning SAG is interesting and would be highly appreciated since this process is tedious and requires numerous measurements.

I will try to implement this feature on my solution, as well as the mobile application representing a practical method to store data on a ride necessitating only the user's phone. However, the string encoder sensor is out of the scope of this thesis due to implementation difficulty and price.

Motions instruments, Motion IQ: The Motion IQ solution, Figure 10 [9], works with a linear potentiometer for recording displacement and velocity. Such sensors are composed of a static main body and a shaft or a mobile section. A resistive element and a wiper carrier connected to the mobile part occupy the body of the sensor. The output voltage is proportional to the position of the wiper on the resistive element [10]. Two different sensors record the variations in suspension length. The fork data are acquired by the fork tracer available in a 200mm or 300mm length, acquiring data at 200Hz [11]. The shock data are acquired by MIPS (Motion Instruments Position System) a 9mm shaft with four measurement lengths from 50mm to 150mm also working on 200Hz [12]. The data are sent to a specific phone app through BLE 4.2. The sensors adopted are high quality and perfect to record linear movement, such as suspension travel. However, among all the proposed solutions, this one is the most expensive and might unsuit some riders' wallets. A better solution should be more affordable.

Figure 10: Motion Instruments solution, © Motion Instruments

BYB Telemetry:

For the BYB Telemetry solution shown in Figure 11, the variation in suspension position is acquired by linear potentiometers, as Motions Instruments' solution. However, their sampling rate is faster, 1000Hz [13] due to their unique acquisition unit placed on the handlebar. The advantage of this sampling rate is a more detailed graphic with possibilities for better suspension analysation, but again the solution is too expensive.

Figure 11: BYB Telemetry solution, © BYB Telemetry

Quarq, Shockwiz:

Figure 12: Quarq Shockwiz Solution, © Quarq

Quarq and RockShox are companies that are owned by the enormous MTB group: Sram. The Shockwiz [14], shown in Figure 12, propose an innovative approach to suspension tuning. Their solution to track fork travel is based on an air pressure sensor monitoring the pressure inside the air chamber. Indeed, the more the fork is compressed, the more pressure is created inside the air chamber. The small and easy-to-mount device is directly connected to the air spring valve and sends the data to their dedicated mobile phone application. Connected to the suspension's air spring, the Quarq acquires suspension data to process and display them as well as offering optimisation suggestions to the user. The application takes into consideration the user ride style for different tuning suggestions (Balanced, playful, stiffness, liveliness). This solution is complete, small, and easy to mount, reasonable in terms of price, but only works for air spring fork and shock.

K. Grund thesis works:

Figure 13: ToF sensor solution, ©K. Grund

In addition to companies, some individuals have already worked on a similar project. Mr K. Grund is one good example with his thesis "IoT Video Telemetry System Implemented on Mountain Bike"[15] in which he created a real-time data overlay of his GoPro footage to display information such as fork and shock compression, acceleration, as well as front and rear brake lever engagement. Even if the final goal differs, for displaying suspension compression on his overlay, he had to assess the same problem I faced: determine the vertical position of the fork.

For this purpose, he decided to manipulate a linear Time of Flight sensor, more precisely an Adafruit VL53L0X. This sensor works in the same way as the ultrasonic sensor; an entity bounces back on an obstacle and the time it takes gives us the distance. Instead of using soundwaves, this Vertical Cavity Surface Emitting Laser (VCSEL) emits an infrared laser at 940 nanometres. According to the datasheet of the VL53L0X [16], this sensor can be configured into four range profiles in case the application needs precise measurement regardless of measurement time (5Hz), or long-ranging (30Hz), or high-speed (50Hz) regardless of the accuracy.

Range Profile	Range timing budget	Typical performance	Typical application
Default mode	30ms	1.2m, accuracy	standard
High accuracy	200ms	1.2m, accuracy < +/- 3%	precise measurement
Long range	33ms	2m, accuracy	long ranging, only for dark conditions (no IR)
High speed	20ms	1.2m, accuracy +/- 5%	high speed where accuracy is not priority

Table 2: VL53L0X range mode, © STMicroelectronics

The laser beam grants its best precision when it reflects on a white surface. In contrast, the worst precision is achieved on black surfaces due to the property of a black body to retain radiation, as depicted in Planck's Law. Besides, precision can decrease outdoors due to the collector cone absorbing unwanted infrared from the sunlight.

	Indoor (no infrared)			Outdoor		
Target reflectance level (Full FOV)	Distance	33ms	66ms	Distance	33ms	66ms
White Target (88%)	at 120cm	4%	3%	at 60cm	7%	6%
Grey Target (17%)	at 70cm	7%	6%	at 40cm	12%	9%

Table 3: VL53L0X precision, ©STMicroelectronics

D. Longo motorbike telemetry:

Figure 14: Ultrasonic sensor fork mount, © D. Longo

The idea behind the motorbike telemetry project of Mr D. Longo was to create a telemetry system working with Arduino and Azure, inspired by moto GP race TV telemetry images. The data acquired, among others are speed, GPS, lean angle, wheelie angle, G acceleration, tyre temperature, front, and back shock absorber compression. The data are sampled at $2Hz$ which in our case is insufficient to precisely track fork travel, but enough for its application. The sensor employed to track compression of the shock absorber is an ultrasonic sensor mounted on the fork facing the front fender, Figure 14. An HC-05 Bluetooth module then sends the data.

An ultrasonic sensor emits sound behind human ear perception [17], at $40KHz$. Through its transmitter, the sound waves travel through the air in search of an obstacle to bounce back on. Once an object is detected, the reflected waves are

redirected to the receiver where the distance is computed based on travel time and the speed sound constant $340\ m/s$.

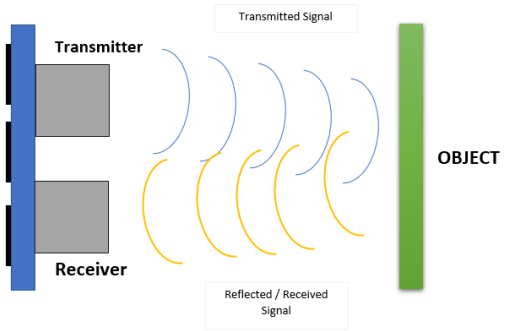


Figure 15: Ultrasonic sensor functioning, ©Ansari Aquib

Andrextr works on rear suspension telemetry & data acquisition:

André offers mountain bike analysis focused on design and kinematics of the rear suspension, including analysis of anti-squat, anti-rise, chain-growth, axle path and leverage ratio [18]. He brings to the fore a rear suspension analysis based on GoPro footage. In his setup, the GoPro records the compression of the shock over time, afterwards, the displacement and velocity of each stroke are extracted frame per frame through a motion analysis software: Tracker [19].

Finally, he plots the results and analyses the comportment of the shock over a small test called the "Curb test" focusing on finding critical shock rebound adjustment. The test consists of him riding down a small curb seated on his bike; the presented Figure 16 [20] showcases the result obtained by the GoPro method.

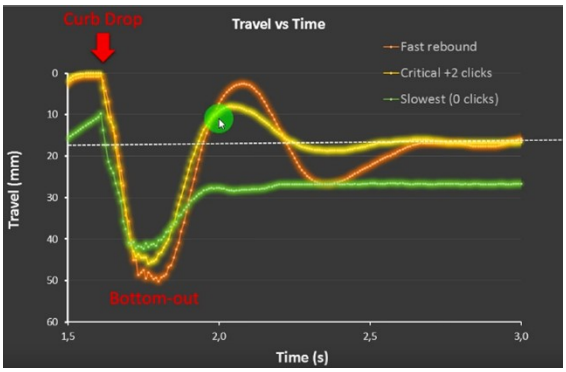


Figure 16: Andrextr curb test result, ©Andrextr

The orange curve is the result of the test with a rebound to fast, creating oscillations, symbolising losing stability since the rider centre of gravity fluctuates more than necessary. In green, the rebound is too slow, causing the shock inability to recover to SAG (represented by the dotted line on the graphics), consequently the shock risks to bottom out over multiple hits. Finally, the yellow curve represents a critical rebound setting, only one overshoot and one oscillation is achieved before returning to the SAG value.

I already decided to work with a GoPro camera before discovering his works. However, I will retain the software motion analysis method used in his project as well as try to adapt the "curb test" from rear suspension to front suspension.

2.3 Hardware

In the previous section, we established that different sensors could handle fork travel tracking.

On one side, the physical sensor with one extremity is fixed to the moving part and the other side to the static part, such as the linear potentiometer and the draw wire, offering better reliability and precision.

On the other side, light or sound-emitting sensors based on reflection, such as the Time-of-Flight sensor using an infrared laser, or ultrasonic sensor, offer an affordable solution.

Aside from those technologies, Quarq presents an innovative air pressure sensor to determine fork travel, and Andrextr works demonstrate the capability of GoPro and motion analysis software to determine fork travel at each video frame rate. Accelerometers could also be considered since a double integration of the acceleration permits to obtain the distance in theory.

2.4 Software

The free-to-use software Kinovea V0.9.3 is a motion analysis software that permits to determine angles, positions, speed of tracked elements of a video. Kinovea is specifically designed to satisfy four missions: observe, capture, measure and annotate sports video. The video is analysed frame per frame and exports the data into a .csv file. The framerate of the recording then delimits the frequency of data acquisition.

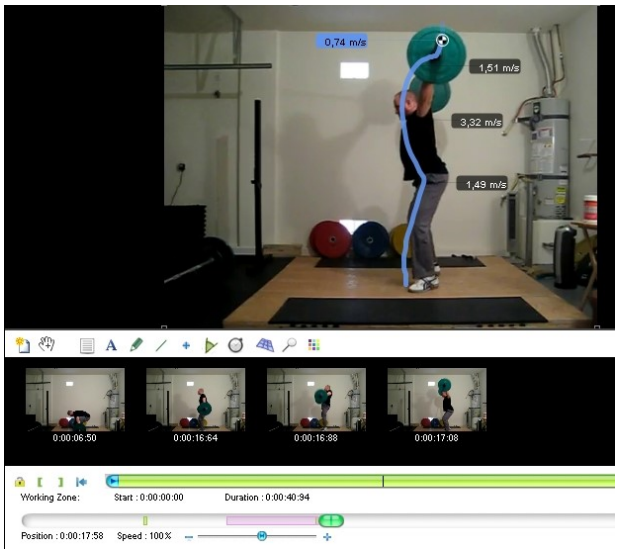


Figure 17: Kinovea tracking example, © Kinovea

To correctly exploit the software and output understandable data, we need to provide a known reference length to convert pixel unit to metric or angular unit. In our case, the reference could be the length of the uncompressed stanchions. This value is easily known by the rider and could be verified by physically measuring it. This value may differ from the amount of travel, for example, my fork is a 150mm travel, but in my personal case, the distance is 160mm long.

The advantages of such software are that the sampling rate is directly correlated with camera frame rate, action camera frame rate is generally high, up to 240Hz for a GoPro Hero 7. In contrast, the drawback of such software might be accuracy. Not only vibration can be induced by the camera mount, but also the video can suffer from radial distortion induced by the camera lens. However, the software can perform a lens calibration in Kinovea.

Efficient tracking relies on video quality and so, mostly about the lightning condition, which is a random factor in outdoor sports.

In addition to Kinovea, Tracker by Physlets is also another free-to-use software solution. Tracker is based on Open-Source Physics (OSP) Java framework [19], while Kinovea is based on OpenCV [21]. Both software will be tested with the same video recording and a comparison will be made in chapter 5 “Video Processing”.

3. Design and Experimental Setup

This part describes the experimental setup: the bike, the trail and obstacles used to carry out the tests as well as filtering the possibility listed in the preceding section and explaining the design of the embedded electronics by providing the goal and expectation of the acquiring unit, as well as wiring schematics and software architecture.

3.1 Mountain Bike Used

The mountain bike I used is called a Hardtail; in contrast to a full-suspension bike, a hardtail misses a rear suspension. The frame is however designed to accept a fork with a large amount of travel (150mm or superior). Figure 18 [22] below provides the geometry of my mountain bike.

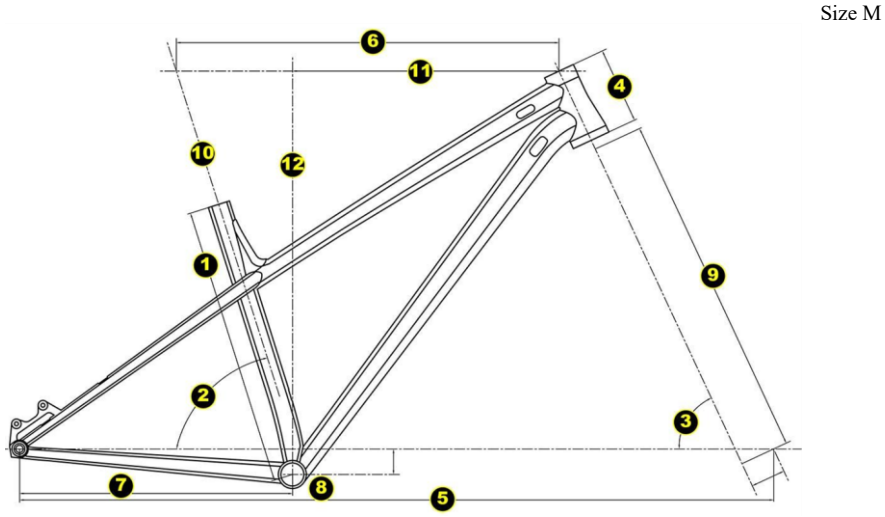


Figure 18: Commencal Meta HT 2021 Geometry, ©Commencal

A geometry chart such as the one above is useful to predict the comportment and handling of the bike. As an example, the more wheelbase a bike possesses, the more stable it will be, but in contrast, the bike will lose in manoeuvrability. It is a crucial point to notice that the wheelbase is dependent on the fork length, in a way that the more the fork compresses into its travel, the more the wheelbase and the stability are reduced. Some components can change the geometry of the bike. Obviously, the fork and its length are dictated by its current travel, but also by the tyres. Installing a bigger tyre on the front and a smaller tyre on the back decreases the head tube angle expanding the wheelbase. The bike I used in this thesis, as seen in Figure 19, is a custom build around a Commencal Meta HT 2021 frame, a Fox 36 grip2 fork, offering 150mm of travel, the front and back tyres are Maxxis designed for 29” wheel with 2.3” section width, thus corresponding to an ETRTO 58-622. ETRTO stands for European Tyre and Rim Technical Organisation and is responsible for the standardization of pneumatic tyres, rims, and valves in Europe.

Figure 19: Bike used

3.2 Unitary Tests and Trail Tests

Before entering those tests, the SAG should be correctly tuned. The obstacle tests present the advantage of being easily repeatable and short, and the same obstacle will be practised for each test. They are designed to follow the correct order of suspension tuning: SAG, rebound (start from fast rebound), LSC, HSC (start from no compression). This tuning order can be explained by the fact that the rebound of the fork mostly depends on spring and rider weight, while compression is more trail dependent. Then the trail tests are here to validate the resulting tuning, assuring it is coherent in a real situation. Tuning is always a compromise to find the balance that suits the rider and the trail. Table 4 [23] lists the comportment of the bike based on too little or too much of each adjustment.

	Too little	**Too much**
LSR	The fork sits low in its travel since it fails to come back to its SAG, traction loss.	Create oscillation, less stable, less pedal efficiency, but feel plush, more traction.
HSR		
LSC	Less pedal efficiency.	Firmness of the suspension, less comfort, but less weight distribution, more stability.
HSC	Bottom-out.	Less sensitivity, feel stiff, harsh, less traction, full travel unused.

Table 4: Compromise of each adjustment

3.2.1 Test on obstacles

- Test 1: Small drop	(T1SD)	- Test 2: Successive hits	(T2SH)
- Test 3: Technical climb	(T3TC)	- Test 4: Jump	(T4J)

T1SD focuses on setting the rebound of the fork. This test consists of riding down one drop of approximately 20cm in a neutral position at medium speed.

Ideal result: before the drop the fork is compressed at SAG value when the front wheel crosses the edge of the drop the fork expands to its full travel. On the impact, the fork compresses and springs back to the SAG value with the minimum overshoot possible and one oscillation at maximum.

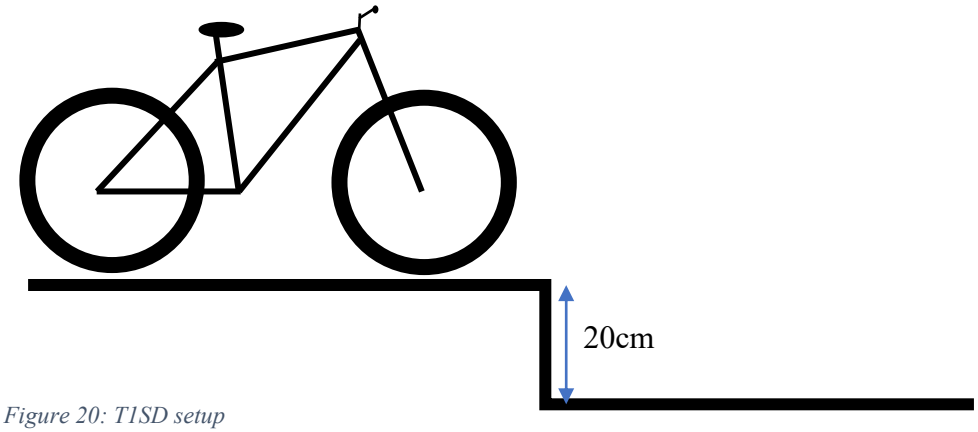


Figure 20: T1SD setup

T2SH goal is to validate the results of the first test by soliciting rebound adjustment in another way. This test consists of riding down successive identical obstacles such as a stair.

The ideal result would be a rebound quick enough to stay high in the fork travel and prevent the fork from bottoming out on small but quick successive hits.

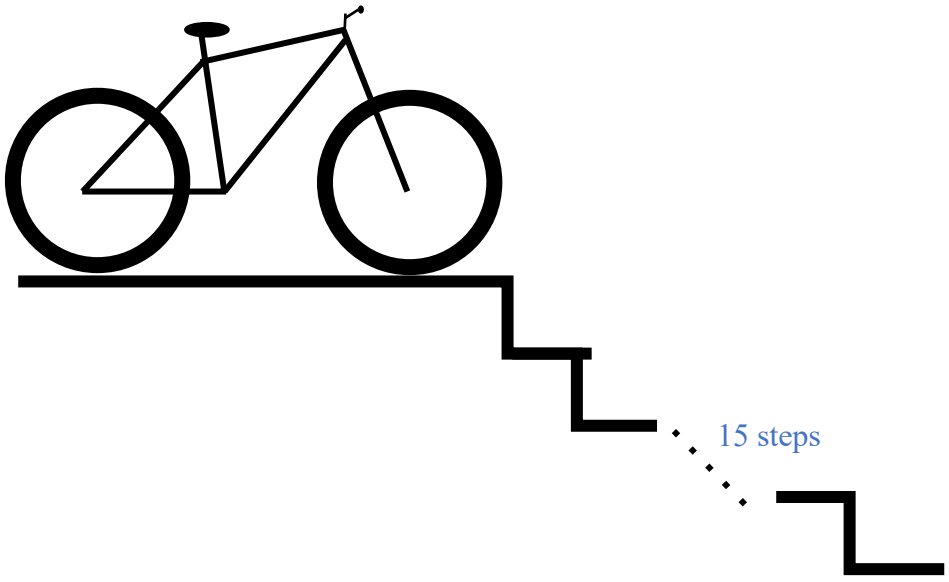


Figure 21: T2SH setup

T3TC targets traction and pedal efficiency offered by rebound setting and low-speed compression. A suitable obstacle can be a short but high climb featuring roots, rocks, or any technical situation where the rider needs to adjust his body position correctly.

Metrics to determine the performance on this test could be the time to achieve the climb or number of attempts before success.

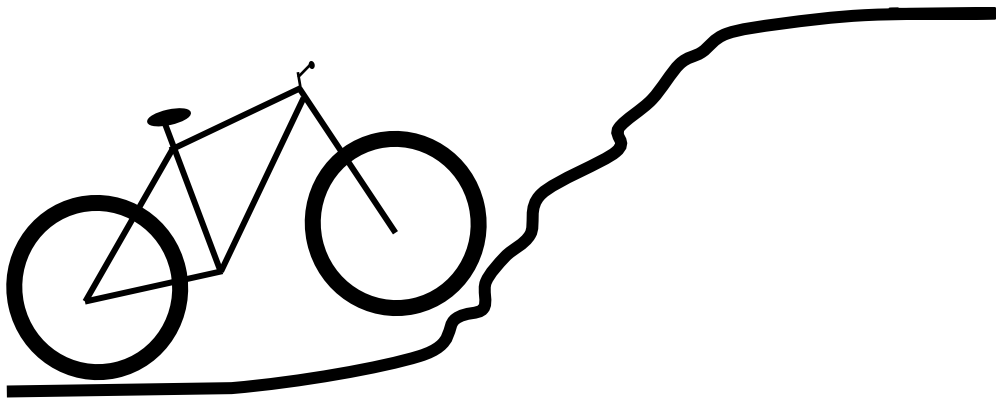


Figure 22: T3TC setup

T4J measures the bottom-out resistance of the fork. A suitable obstacle is a jump that the rider can easily repeat with consistent speed, the size of the jump should depend on what the rider usually rides. The size and the difficulty of the jump should stay in the rider's comfort zone.

The bottom-out resistance can be altered by adding HSC clicks or tokens. For an ideal result, fork travel should remain above the last 15% mm of travel, saving extra absorption capacity for bigger jump, or harsh landing.

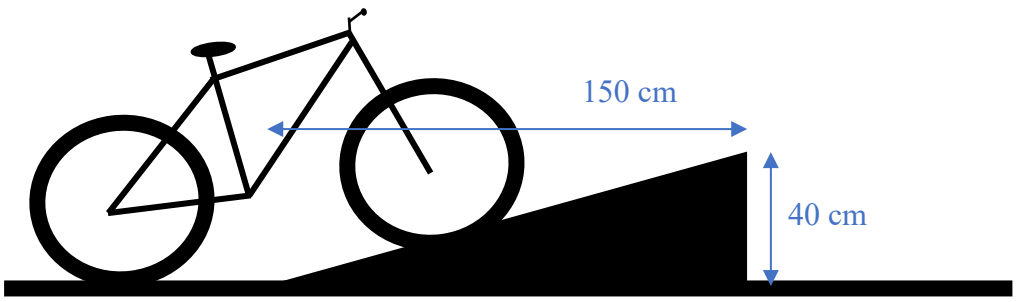


Figure 23: T4J setup

Table 5 below resumes the goals of the tests.

	Focus on	Adjustment targeted
T1SD	Returning to SAG value.	Rebound
T2SH	Staying high in the travel.	Rebound, LSC
T3TC	Traction, pedal efficiency.	LSC, rebound
T4J	Bottom out, bike control.	HSC, HSR

Table 5: Unitary tests goal

3.2.2 Test on a trail

The trail test TT1 is a one-minute-long flowy descent into the wood, 4km from home, perfect for easy repeatability. This short trail (500m) offers a low gradient of -6.2%, one jump, a flat corner, roots, and rocks section. The trail test goal is to verify in a real environment the setting constructed through the previous unitary test.

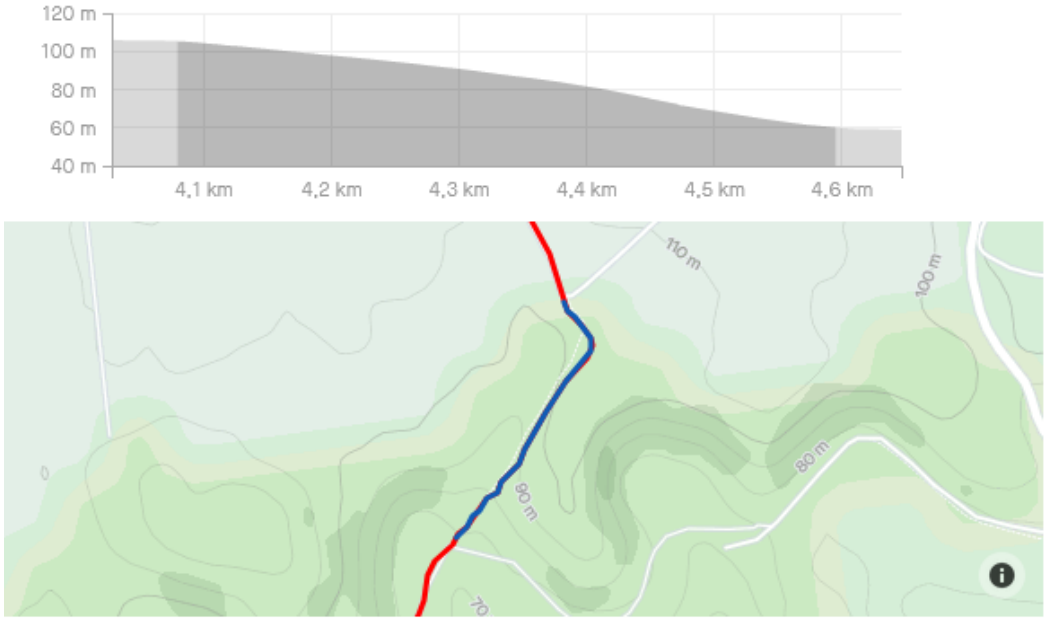


Figure 24: Test Trail topography

The GroPro method is unfortunately unsuitable for the trail test; at some point, due to quick steering change and lean angle, the tracking fails, resulting in unusable data.
A solution would be to create a segmented tracking avoiding elements of the video creating the failure. This solution will result in multiple .csv files to concatenate with no data in between. This solution will elongate the process and is tedious

3.3 Mainboard and Sensor Selection

Now that the bike setup and test have been defined it is time to focus on the electronics that will permit to analyse those tests. This work will be divided into two sections, the first one is consecrated to the build's crucial element: the mainboard, connecting the sensor, the communication device, the battery, and other electronics altogether. The second section is the sensor selection, they are responsible for transcribing real physical input into digital output creating our precious data to analyse.

The few mainboards I have at my disposition are the following:

Board	Flash Memory	SRAM	CPU/ Microcontroller	clock
Arduino Uno	32 KB	2 KB	ATMEGA328P	16 MHz
Arduino Mega	256 KB	8 KB	ATMEGA2560	16 MHz
Raspberry 3B+	-	1 GB	ARM Cortex-A53	1.4GHz

Table 6: Mainboard specifications

I choose an Arduino board for its simplicity, low price, and the presence of libraries dedicated to the available sensors. The Raspberry seems too excessive for this purpose.

I decided to work with an Arduino Uno board for its small size, the software and the components can be easily switched from a board to another in case of failure or switch to the Mega board if memory space is insufficient for the libraries. Also, I am unsure of the Raspberry efficiency with a 9V 650mAh power supply.

The sensors for the data acquisition are composed, on one side, of the embedded electronics sensors such as ultrasonic ranging, Time of Flight, and accelerometers.

On the other side, the internal GoPro's sensor includes a gyroscope, accelerometer, and a GPS module. The data resulting from those sensors are stored in the video file metadata and can be extracted [24].

Physical measurement sensors such as draw-wire or linear potentiometer offer the most accurate value since they are protected from external perturbation and operate at a higher sampling rate than light-emitting or sound-emitting sensors due to their intrinsic technology (bounce-back time). However, I choose to avoid working with those physical measurement sensors due to their high price.

The sensors list is now composed of:

- Ultrasonic ranging sensor
- Time of Flight sensor
- Two accelerometers

Those three different technologies will be put operational in terms of hardware and software in the following sections: tests will be performed, and a discussion will take place to determine whether those sensors are suitable for this project.

Table 7 below illustrates this list of sensors with their noticeable characteristics.

	Sensor type	**Com.**	**Module**	**Range**	**Max update rate**	**Address**
Embedded	Ultrasonic ranging [25]	Digital	SR04-HC	2cm-4m	40Hz	-
	Time of Flight [26]	I2C	VL53L0X	30mm-1m	60Hz	0x29
	Accelerometer1 [27]	I2C	LSM6DSOX	± 2/4/8/16g	6.7KHz	0x6A
	Accelerometer2	I2C	LSM6DSOX	± 2/4/8/16g	6.7KHz	0x6B
GoPro [24]	3-axis gyro				400 Hz	
	3-axis accelerometer				200 Hz	
	GPS position (lat/lon/alt/spd)				18 Hz	

Table 7: Sensors specifications

To offer feedback to the user and debug ability, a 0.96” OLED I2C screen will be implemented on the handlebar. Its small size will not interfere with the cockpit nor obstruct the vision of the trail and will be perfect for displaying the status of the electronics, battery level, and SAG setting.

Concerning Energy supply, a first test will be performed with a 9V 650mAh battery to estimate the power consumption of the embedded electronic. This battery is removable, rechargeable, and compact. The power battery level is a critical characteristic. The software needs to integrate an interruption if the battery level is below a determined limit to save the result and to prevent data of a full downhill run to be lost forever. Another danger is, if the circuit shut down while the SD card file was incorrectly closed in software, it could lead to SD card destruction.

3.4 Hardware diagram

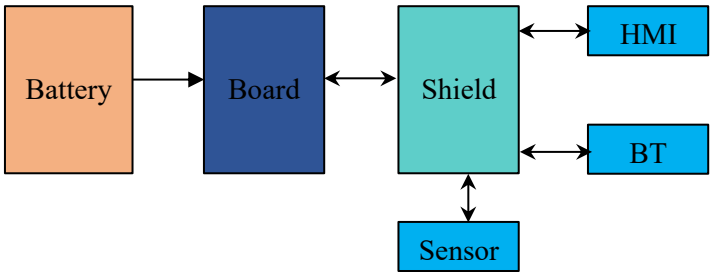


Figure 25: Hardware Diagram

Only the battery is connected to the Arduino board through the 9V jack plug, all the other components: sensor, Bluetooth and screen are firstly connected to a shield designed for the board, thus increasing flexibility and board switch.

3.5 Sensors Mounting

The sensors will be mounted with zip ties, along with a small part of an inner tube placed between the sensors and the fork. The inner tube will prevent unwanted electrical conductibility, protecting the sensor from possible short-circuit, as well as protecting the fork while procuring a strong and grippy mount.

One accelerometer is inadequate to determine fork travel. Indeed, trail elevation or rider lifting the front wheel can interfere with the correctness of the measurement. We will obtain the fork travel by computing the value difference of the two sensors, one mounted on the lower legs while the second on top of the fork stanchions.

The sonar and time of flight sensor will benefit from the same design; the sensor will be mounted above the stanchions and a reflective piece on the lower legs. The reflective piece needs to be white, Adafruit states that white colour delivers the best reflection and more accurate value of the ToF sensor. However, the size needs to be taken into consideration; too small, and the sonar will incorrectly reflect on, too large, and it will impact riding.

The Arduino board, the power supply, and the other electronic components will be mounted to the handlebar inside a plastic box drilled to accept two zip ties securing it in two points of the handlebar, refer to Figure 26. The outside sensors will be wired to the mainboard with a JST XH connector at both extremities to facilitate removal, while keeping a strong connection, unlike the Dupont connector. The Bluetooth module is also connected to the board with a JST XH connector because it uses the TX and RX pin of the Arduino board, the same that operated for serial communication while uploading a sketch. When a new sketch is uploaded, we can directly unplug the connector instead of de-soldering the module. This box is non-waterproof but can ensure dirt and dust protection as well as being easily removable with the zip ties. The Arduino board is held by three nylon screws to prevent

unexpected shortcuts. The screws also elevated the board for the zip ties. The position of the box on the handlebar does not obstruct rider vision, permits good Oled screen visibility, has good support on the stem to prevent the box from rotating. This position also offers protection in case of a crash.

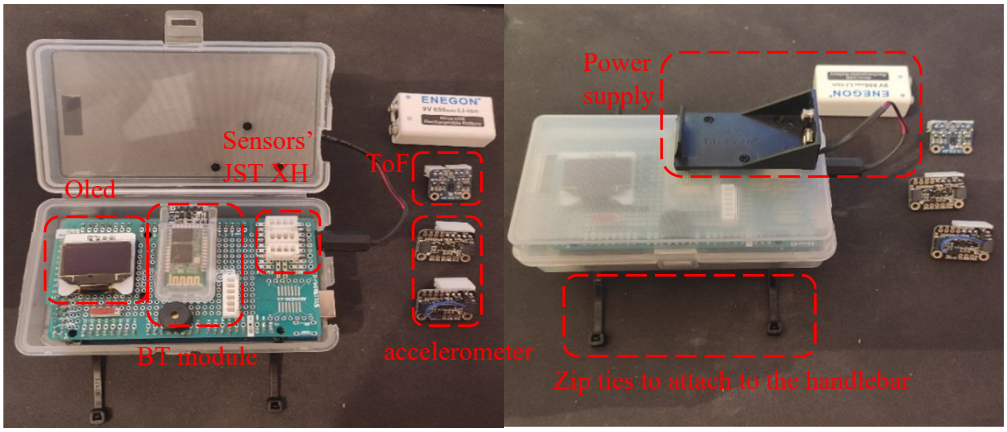


Figure 26: Embedded electronics setup

The GoPro Hero 7 Black editions will be mounted on the side of the frame, on the front triangle, with two zip ties, a rubber inner tube to prevent damaging the frame and procuring grip and a piece of high-density foam to apply even pressure.

Figure 27: Gopro mounting

For this application, we will exploit Hypersmooth, the GoPro stability technology. Hypersmooth is an electronic image stabilisation (IES) that will help filter any unwanted vibration due to the mount solution. The GoPro will be oriented in a way to view the back of the fork, with the lens parallel to the stanchions to obtain better results in object-size determination.

3.6 Embedded Software

The Arduino solution covers various problems: acquiring a distance, communicating via Bluetooth, Human-machine interface. To ensure the entirety of the design works properly, it is primordial to break it down into smaller parts and verify their integration. For each part, both a different sketch is used containing only the actual code for this task, and a different wiring schematic.

3.6.1 Acquiring distance

- With ToF sensor

- Power source:
  The sensor is alimented by a +5V in its Vin port (red wire) and grounded to its GND port (black wire)

- Communication:
  The communication is based on I2C protocol, the sensor's SCL and SDA ports are respectively connected to the board's SCL and SDA ports.
  (SCL: yellow wire, SDA: purple wire)

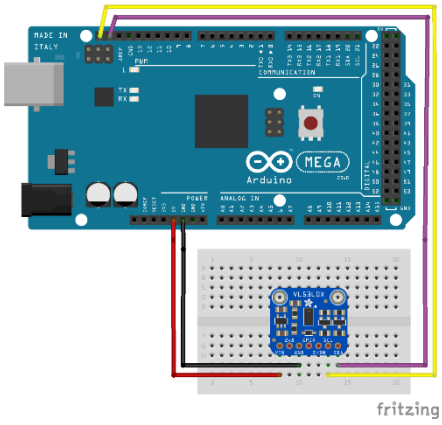


Figure 28: ToF wiring

test_distance §

```
#include <Wire.h>                                   // Library for I2C protocol
#include "Adafruit_VL53L0X.h"                       // Library for the ToF (Time of Flight) sensor
Adafruit_VL53L0X    tof = Adafruit_VL53L0X();       // Declaring an object to use the Tof sensor library

void setup() {
  Serial.begin(115200);                             // Open serial port, set data rate at 115200 bits/sec
  if (!tof.begin()){                                // Check if the Time of Flight sensor is present
    Serial.println(F("Failed to boot VL53L0X"));
    while(1);
  }
}

void loop() {
Serial.print("Distance (mm): ");
VL53L0X_RangingMeasurementData_t measure;
Serial.println(measure.RangeMilliMeter);          // Convert the distance in mm
delay(1000);
}
```

Figure 29: ToF Code

The specific Adafruit library is used to compute the distance in the setup loop. Before acquiring the distance the code certifies that the sensor is correctly detected. Any error at this step could indicate an incorrect connection between the board and the sensor, or that the sensor itself is malfunctioning.

- With ultrasonic

- Power source:
  Arduino's +5V connected to VCC (red wire)
  Arduino ground connected to GND (black wire)

- Communication:
  D2 to trigger pin (blue wire)
  D3 to echo pin (teal wire)

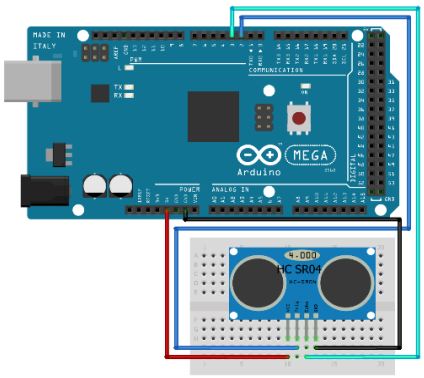


Figure 30: HC-04 wiring

test-HC-SR04

```
const byte TRIGGER_PIN = 2;
const byte ECHO_PIN = 3;
const unsigned long MEASURE_TIMEOUT = 25000UL; // timeout 25ms = ~8m à 340m/s
const float SOUND_SPEED = 340.0 / 1000;

void setup() {
  Serial.begin(115200);
  pinMode(TRIGGER_PIN, OUTPUT);
  digitalWrite(TRIGGER_PIN, LOW);
  pinMode(ECHO_PIN, INPUT);
}

void loop() {
  digitalWrite(TRIGGER_PIN, HIGH); //send an HIGH impulse of 10µs
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);
  long measure = pulseIn(ECHO_PIN, HIGH, MEASURE_TIMEOUT); //measure time between the impulse and the echo
  float distance_mm = measure / 2.0 * SOUND_SPEED;         //compute the distance
  Serial.println(distance_mm);
  delay(40);
}
```

Figure 31: HC-04 code

The trigger pin of the sonar is set to HIGH for 10µs and then is switched back to LOW to create an ultrasonic impulse, then the code measures the time the impulse takes to bounce back on the obstacle and reach the echo pin, once the time is known the code computes the distance with the basic formula:

$$d = v \times t$$

with the velocity v representing the speed of sound constant, and t the time measure that needs to be divided by two because it represents the time needed to achieve two times the desired distance due to the bounce back.

The distance is then printed in mm to the serial.

Figure 32: HC-04 result

3.6.2 HMI

The goal of this test is to test the I2C Oled screen by displaying some text and cleaning the screen after a certain amount of time.

To drive the screen, we use the U8g library. When creating the object, we select the proper screen resolution and driver to match our hardware, and inside the loop, we can add the text to display.

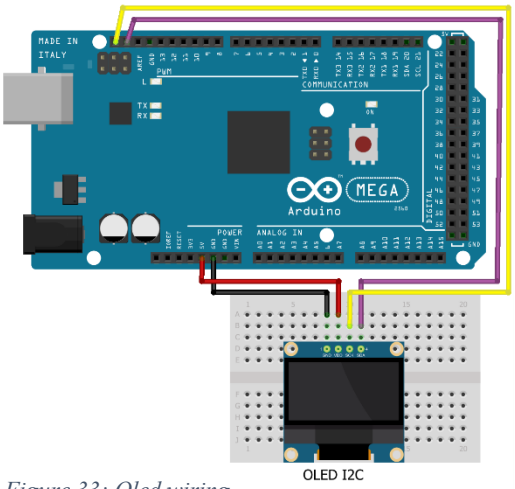


Figure 33: Oled wiring

HMI_TEST §

```
#include <Wire.h>                                  // Library for I2C protocol
#include "U8glib.h"                                // Library for the OLED Screen

U8GLIB_SSD1306_128X32   oled(U8G_I2C_OPT_NONE);    // Declaring an object for using our
                                                   // I2C 128x32 OLED screen with SSD1306 driver

void setup() {
 Serial.begin(115200);    // Open serial port, set data rate at 115200 bits/sec
}

void loop() {
  oled.firstPage();
    do {
      oled.setFont(u8g_font_unifont);   // Select the font of the text
      oled.setPrintPos(0, 10);          // Select the position of the first line (x,y)
      oled.print("SAG: 15%");           // Display a SAG value in percentage
    } while( oled.nextPage() );

    delay(5000);

    oled.firstPage();                   //Clean screen
    do {} while( oled.nextPage() );
}
```

Figure 34: Oled code

The U8G library is used to drive the Oled screen, during the declaration of the Oled object, the screen's resolution (128x32) and communication protocol need to be specified.

The position of the text can be chosen by a pixel coordinate (x-axis, y-axis) with (0,0) representing the top left corner of the screen.

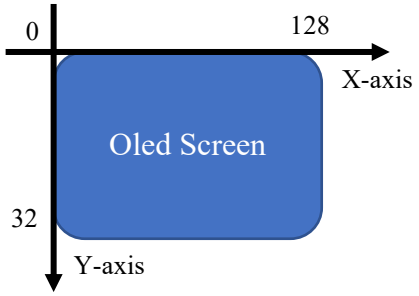


Figure 35: Oled coordinate reference

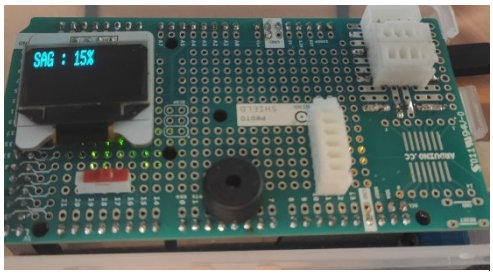


Figure 36: Oled result

3.6.3 Bluetooth module communication test

The goal of this code and hardware is to validate a Bluetooth communication between an Android phone and the Arduino serial monitor. The android phone is equipped with the application "Bluetooth Terminal" from Qwerty [28]. The Arduino Uno board only possesses one set of RX and TX ports already used by the Arduino serial monitor. To prevent this to be an issue SoftwareSerial library is used to emulate virtual RX and TX port for Bluetooth communication. However, this library performs poorly for baud rate over 9600 as well as being incompatible with the Arduino Mega board we finally decided to use. Fortunately, the Arduino Mega board possesses four sets of RX and TX port; we can then use one set for serial monitor and one set for Bluetooth communication.

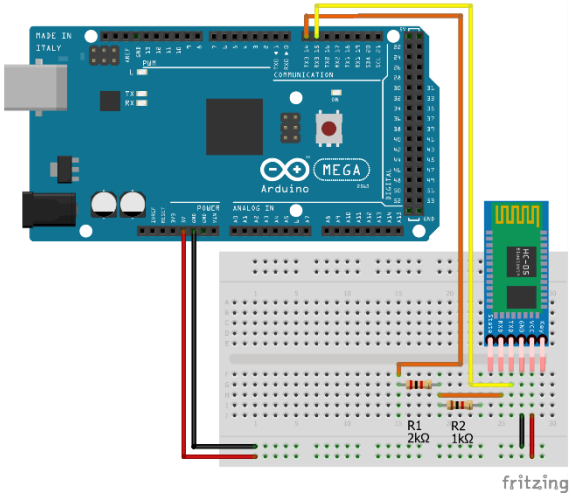


Figure 37: BT wiring

The code is the following:

HC-05-COMMUNICATION-TEST §

```
1  void setup()
2  {
3    Serial.begin(9600);   // open the serial port
4    Serial3.begin(9600);  // open the bluetooth serial port, we use serial3 because we plug our module to TX3 and RX3 but
5                          // we can use serial 1 or 2 however not 0 since it is used for the arduino serial monitor communication
6  }
7
8  void loop()
9  {
10   // send from serial to bluetooth
11   if(Serial3.available()){
12     Serial.println(Serial3.readString());
13   }
14
15   // send from bluetooth to serial
16   if(Serial.available()){
17     Serial3.println(Serial.readString());
18   }
19 }
```

Figure 38: BT code

Serial corresponding to ports RX0 and TX0 is open to receive or send data through the Arduino serial monitor. Similarly, Serial3 corresponding to ports RX3 and TX3 are open to receive or send data. Once the ports are open, the software checks if the user wants to send data from serial to Bluetooth or if the user is receiving data from Bluetooth to serial and wants to display it.

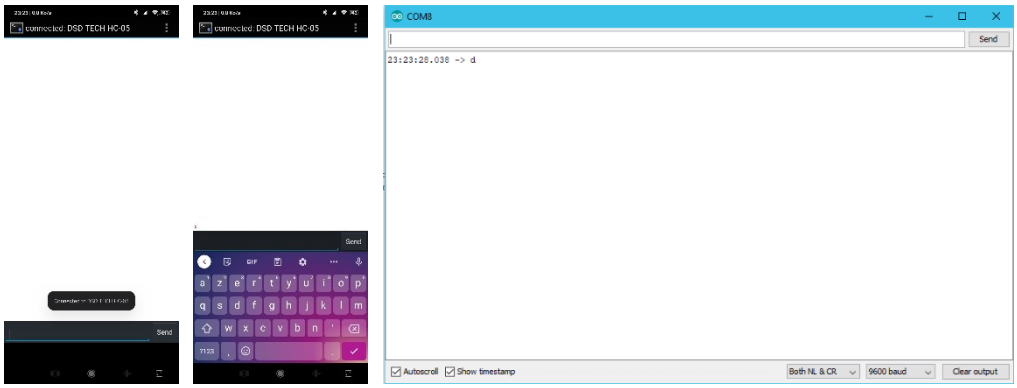


Figure 39: BT result 1

After the code is successfully uploaded into the Arduino board, we open the Bluetooth terminal application, select "Connect a device – Insecure" and send a letter, for example, "d", shortly after we receive the letter "d" on the serial monitor.

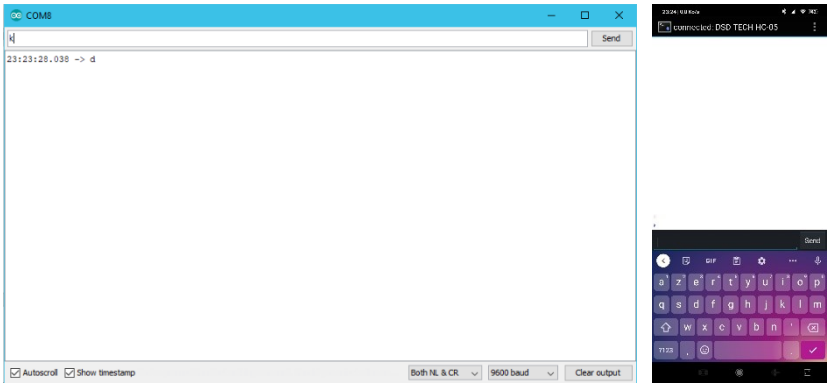


Figure 40: BT result 2

Now, from the serial monitor, we enter the letter "k" and shortly after we correctly receive the letter on the phone Bluetooth Terminal application, thus validating the design of a two-way Bluetooth communication between Arduino and Android phone.

3.6.4 Bluetooth AT commands

AT commands are a set of instructions for setting a module Bluetooth, including visualising and changing password, name, baud rate, role (0 slave, 1 master, 2 slave loop role) of the module. According to DSD Tech, the manufacturer of the Bluetooth module, we need to set the serial monitor to 38400 baud in AT mode.

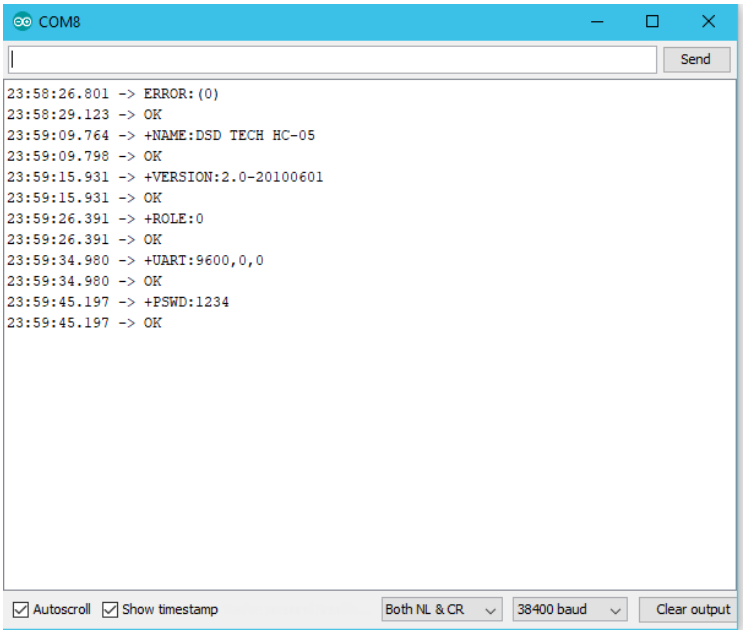


Figure 41: AT commands

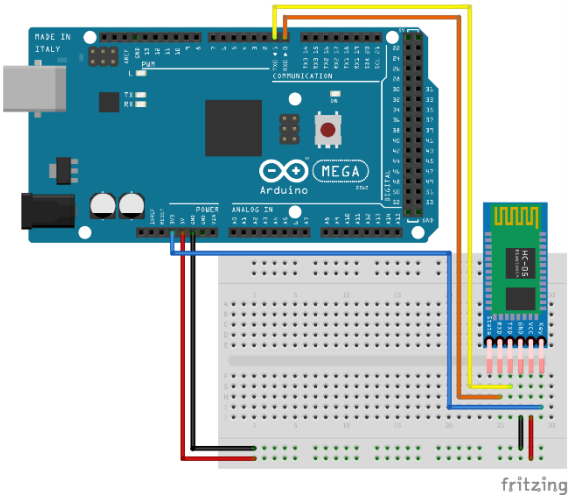


Figure 42: AT mode wiring

To enter the AT mode, only special wiring is needed: the EN/KEY connection of the Bluetooth module needs to be in the HIGH state by providing 3.3V to it, for example, respectively the RX and the TX of the module go to the RX and the TX of the board. A red LED blinking slowly at 2s intervals indicates that the user is in AT mode.

The name of the Bluetooth module, the version, the baud rate, and other specifications have been visualised and could be changed, validating this aspect of the Arduino solution.

3.7 Final Embedded Electronic SF and HW

3.7.1 Goal

As shown in chapter 2, Motions instruments choose a sample rate of $200Hz$ and $1KHz$ for BYB Telemetry. Such rates are unachievable with most of the chosen sensors. The ToF will work at a sampling rate of $40Hz$, and we will determine if this is sufficient for our purpose. Our goal now is to connect via Bluetooth to our Android application, calibrate the sensor, acquire the displacement of the fork, and send this data to our application.

3.7.2 Data type

Table 8 below defined the notation I will use for the variables as well as their values and size.

	Min value	Max value	Bytes
`Boolean`	$false$ (0)	$true$ (1)	1
`char`	-128	127	1
`int8_t`	-128	127	1
`uint8_t`	0	255	1
`int16_t`	$-32\,768$	32 767	2
`uint16_t`	0	65 535	2
`int32_t`	$-2\,147\,483\,648$	2 147 483 647	4
`uint32_t`	0	4 294 967 295	4
`int64_t`	$-9\,223\,372\,036\,854\,775\,808$	9 223 372 036 854 775 807	8
`uint64_t`	0	18 446 744 073 709 554 615	8
`float`	$-3.4028235\ E38$	$3.4028235\ E38$	4
`string`			$n\ char + 1$

Table 8: Arduino data type

Float and double on ATMEGA based boards present no difference [29].

3.7.3 Software Architecture

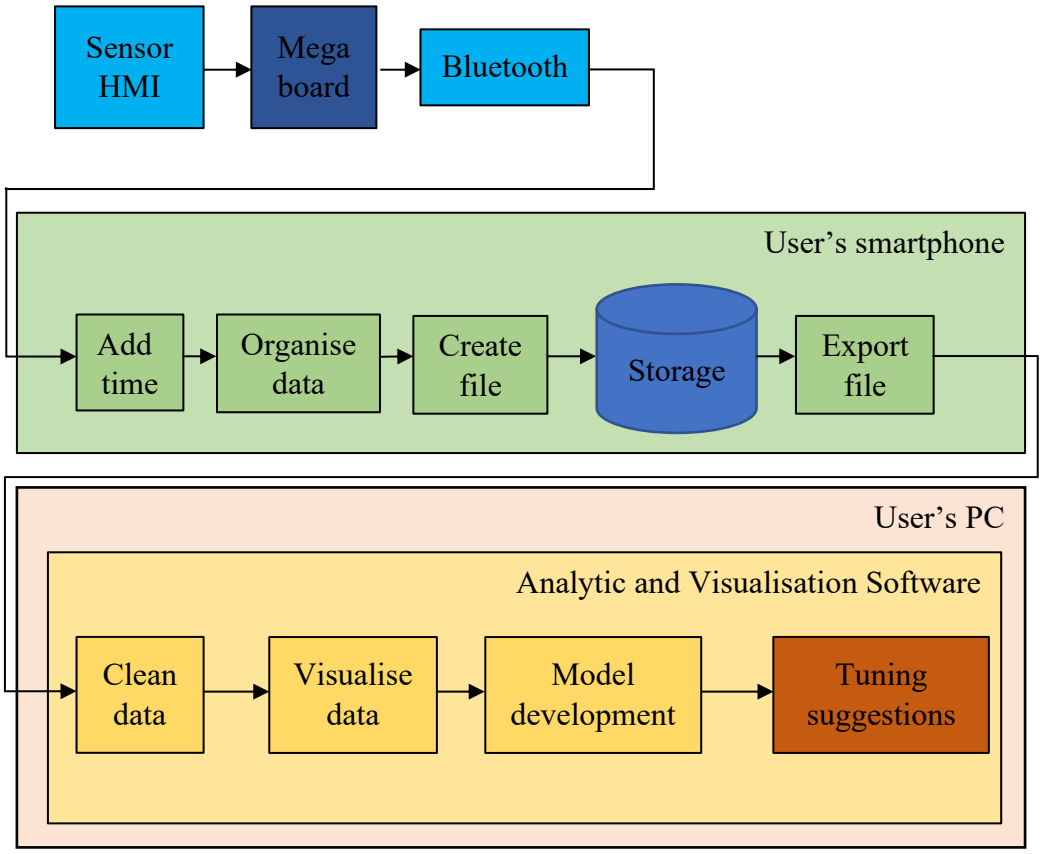


Figure 43: Embedded electronics software diagram

The architecture is divided into three clusters; the first is the embedded environment containing the sensor ordered by the Arduino board communicating via Bluetooth to the second cluster, the smartphone environment. The second cluster role is to date, organise, and store the data acquired on the trail conveniently and automatically. After the trail completion, the rider can transfer the file from its smartphone to the third cluster which is its computer. On this third cluster, the user will execute a program performing data cleaning and processing for the user to visualise the data as well as the tuning suggestions.

3.7.4 Wiring Schematics

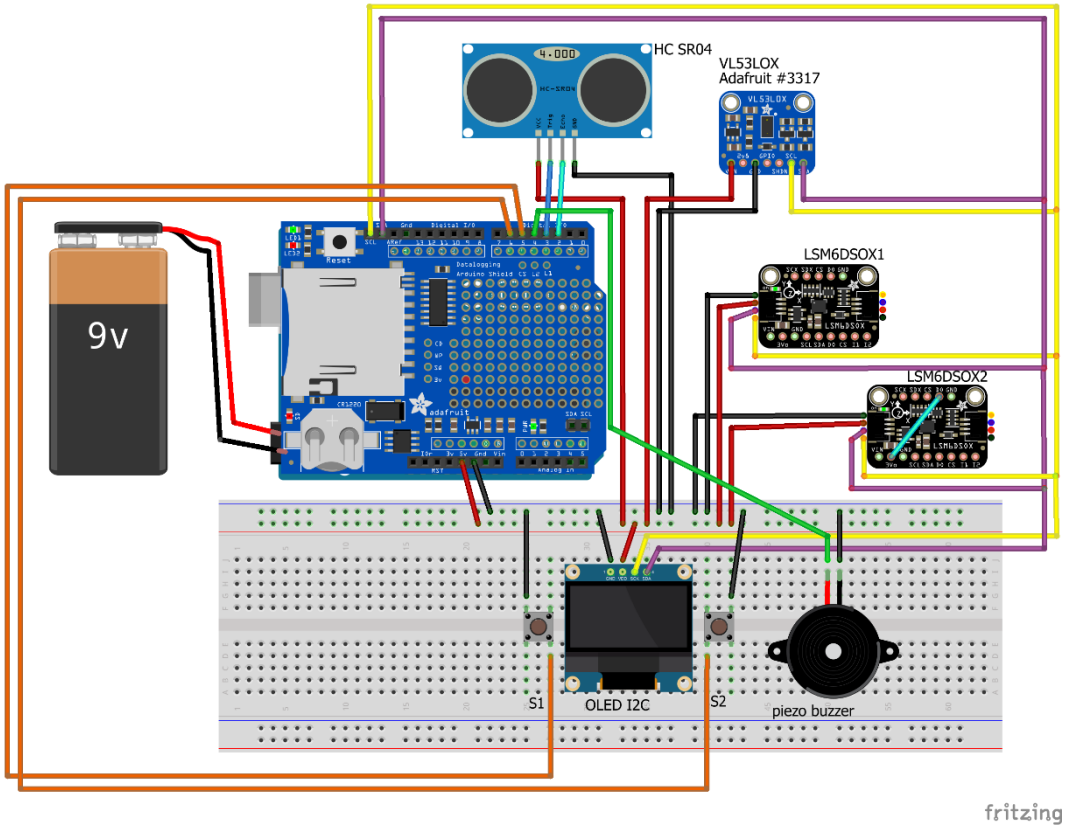


Figure 44: Embedded electronic Arduino Uno wiring

Red wire: +5V
Black wire: GND
Purple wire: SDA line
Yellow wire: SCL line

S1 is the push button that enables the user to select the modes: SAG tuning, and data acquiring. S1 is connected to the digital pin 5.

S2 is the push button to start and stop the acquisition.

The piezo buzzer will create a sound to notify the user of the first and last acquisition, and it will also help with GoPro footage synchronisation. The piezo buzzer is connected to the digital pin 6.

To help reduce the number of components to solder onto the board, we benefit from the internal pull-up resistor for the switches to accurately detect status change.

The first accelerometer LSM6DSOX1 operates at the IIC address 0x6A. The second accelerometer LSM6DSOX2 address is set to 0x6B with the help of the cyan wire between D0 and 3.3V.

The Oled screen, as well as the distance sensors, except the digital HC-SR04, use a common SCL and SDA line for IIC communication protocol.

The Adafruit data logger shield provides RTC and SD card reading through the SPI communication protocol.

The power supply will be managed by a 9V rechargeable battery connected to the 9V jack on the Arduino board.

Unfortunately, only including the needed library resulted in memory leakage. The 32KB flash memory board was insufficient and, therefore, I upgraded the board to an Arduino Mega 2560 offering 256KB of flash memory. The new wiring can be found below in Figure 45.

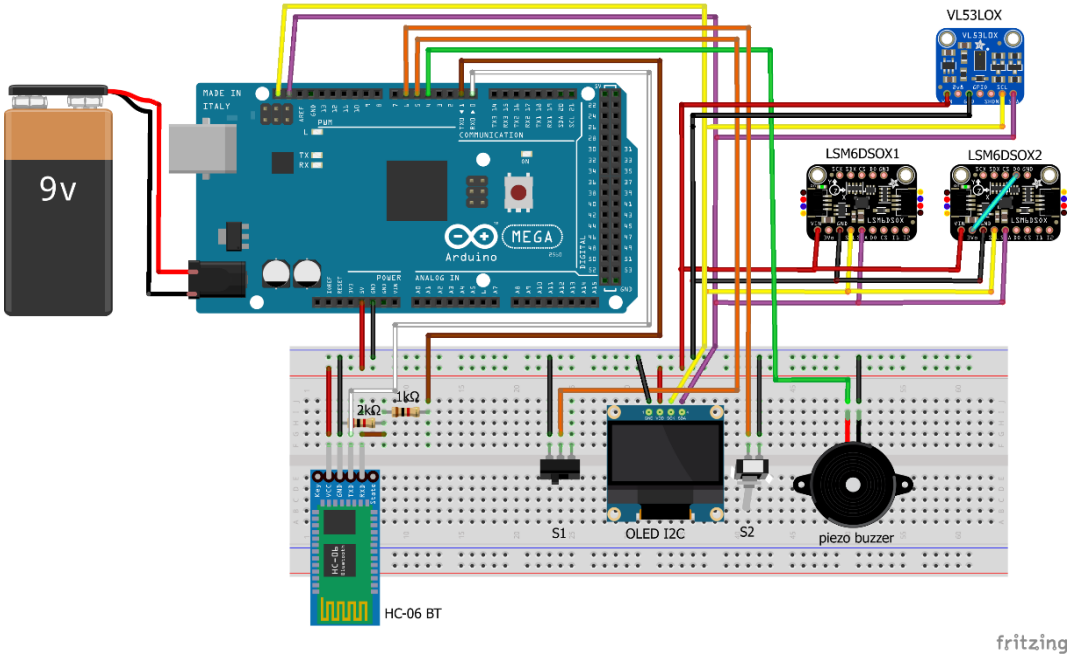
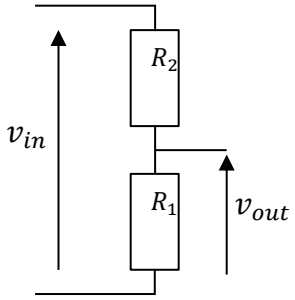


Figure 45: Embedded electronic Arduino Mega wiring

The colour code and components are still the same except for the Adafruit shield that was only compatible with the Arduino Uno board. Being unable to use this shield on the new Arduino Mega means that the SD card reader and RTC are now unusable. An alternative method to provide storage and dating is to transfer the data to an Android application via Bluetooth to the user's smartphone storage. On the data arrival, the time of the smartphone can serve as a reference to the acquisition dating.

The Bluetooth module selected is an HC-05 that can be configured to baud rate up to 115200 matching the baud rate of our Arduino sketch, and can work as a slave and as a master. However, the HC-05 is unsuitable for Bluetooth communication with Apple devices, thus explaining the choice to create an Android app. This module works with $3.3V$, but can be supplied with a voltage between 3.6 and $6V$ on Vin. The only voltage problem is in the RX pin of the HC-05, the Arduino pin delivers $0V$ in LOW and $5V$ in HIGH; thus, we need to design a voltage divider.

The following formula can express the voltage divider:



$\frac{V_{out}}{V_{in}} = \frac{R_1}{R_1+R_2}$ With: $V_{out} = 3.3V$ and $V_{in} = 5V$

A pair of common resistors verifying this equation is:
$R_1 = 2k\Omega$
$R_2 = 1k\Omega$

Figure 46: Voltage divider

4. Android Application

This chapter outlines the goal of the Android application. A step-by-step code breakdown is performed in the same way as the embedded code in the previous chapter. This section also describes which tools were used, their restriction and lists all the ideas alongside the application development.

4.1 MIT App Inventor

App Inventor [30] was developed by Google and are now maintained by the Massachusetts Institute of Technology. App Inventor permits creating an Android application intuitively and graphically; indeed, lines of code are replaced by graphical blocks similarly to LabView by National Instruments. The advantage for me to use this tool is to develop and test more rapidly the features of the application regardless of the aesthetic, and the advantage for the thesis itself is a more comprehensive and compact software to present how the application works.

4.2 Connection to Bluetooth Device

The goal of the application is to receive the data and store the data to easily transfer them to the computer, where they will be analysed. The first subjacent goal is to be able to communicate with our sensor; in other words, create a Bluetooth connection between the phone and the Arduino.

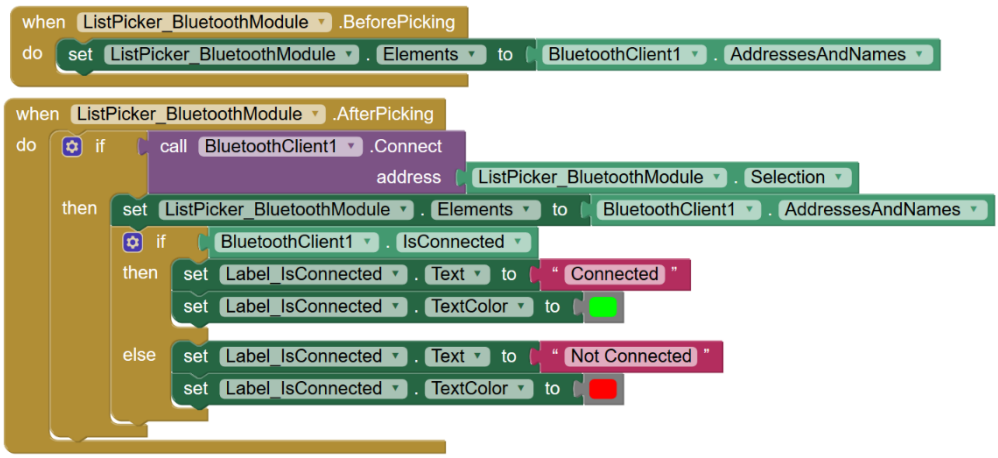


Figure 47: App BT connection

The block at the top of Figure 47 permits us to build the list of all Bluetooth devices paired to our smartphone. This list is then used to display their name and MAC address when the user wants to select the HC-05 module.

The block on the bottom tries to connect to the selected Bluetooth. If it succeeds, the user is informed by the label “Connected” display in green; if it fails, the label displays “Not Connected” in red.

4.3 Receiving data

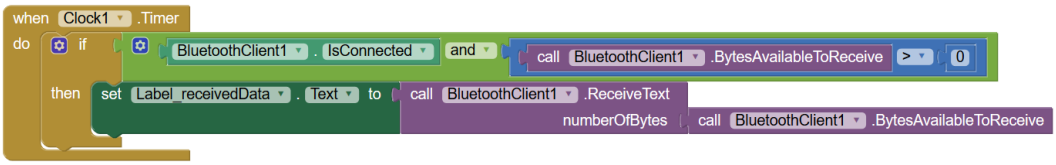


Figure 48: App receiving data

Once the connection is established, the Arduino will send the sensors' data; we need to receive them all. To successfully catch all information, Clock1 is set to zero seconds, meaning we receive information at a rate defined by the Arduino. This block also permits to show the incoming data inside a label to a debug or a verification purpose.

4.4 Save Into File

Finally, the last step for the application is to save the data into a file.

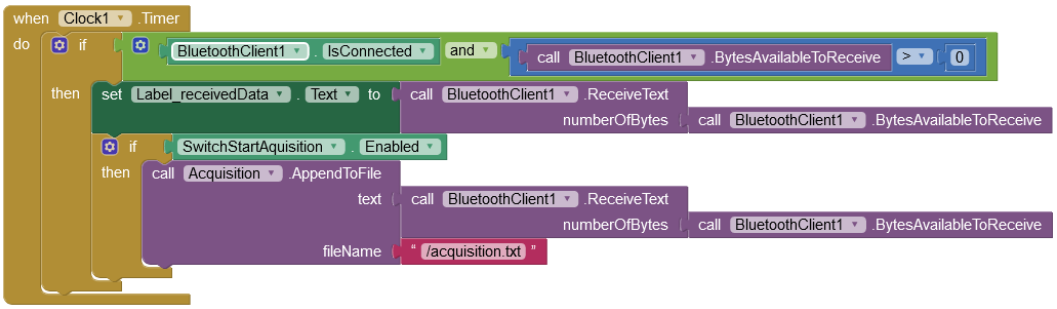


Figure 49: App storage

To offer this possibility, we need to add more material to our previous receiving data block. After displaying the data on the text area, we check if the start acquisition slider switch is enabled; if true, we create an "acquisition.txt" file on the external storage of the phone (on the SD card), containing the received data.

4.5 Application Design

The application with all the components is depicted in Figure 50. We can unanimously state that the resulting application is aesthetically unattractive, but It prioritises functionality and performances over the design.

The list of paired Bluetooth devices is accessible by clicking the top blue button. After selecting the Bluetooth, the label will change its text and colour, and when the Arduino is ready, the data acquisition slider is enabled.

The upcoming data will appear on the central text area as well as being saved on the phone memory card.

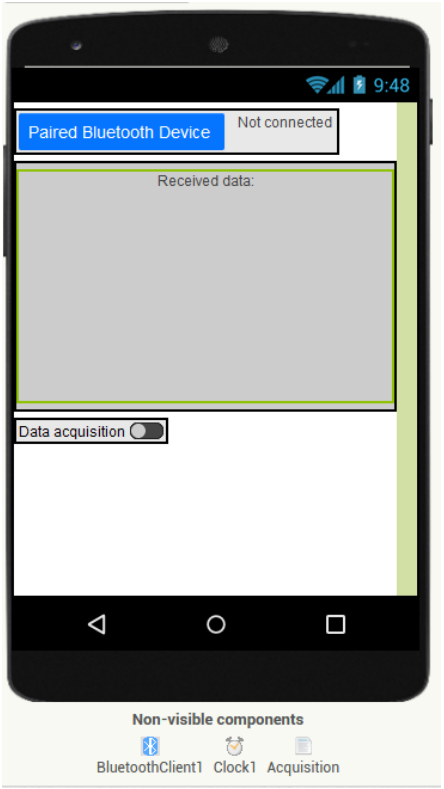


Figure 50: App design

5. Video Processing

This section demonstrates how to manipulate a GoPro to perform telemetry acquisition, from the GoPro settings to the result of the two different software Kinovea and Tracker.

5.1 GoPro settings

The video quality is primordial for the software to perform at its best. The GoPro will record at a standard full HD 1920x1080 resolution, with a framerate of 120FPS benefiting from the standard stabilisation. The Hypersmooth stabilisation would have been a must to filter the undesirable vibration induced by the mount, but the GoPro Hero 7 possesses Hypersmooth only for 60FPS, which results in half times less sample than the standard stabilisation. 240FPS is also an option, but in this configuration, the video quality is unreliable, no stabilisation is offered, and the file size increases considerably, thus increasing processing time.

Table 9 below resume the framerate and stabilisation available on the GoPro Hero 7 Black edition:

Framerate	Highest Stabilisation Available
60	Hypersmooth
120	Standard
240	None

Table 9: Framerate and stabilisation

Finally, the last important parameter is the Field of View. GoPro offers three FOV modes: linear, large and superview. Superview is the widest option ideal for capturing large scenery but possesses strong lens deformation, meaning that a straight line will appear curved on the side of the frame. This is inadequate for our purpose since it could trouble the precision of the software. I chose to work with linear FOV with almost zero lens deformation, presenting no disadvantages for capturing the crucial elements, since the left stanchion and crown could fit in the frame.

Figure 51 depicts the difference between FOV, the left picture is taken with a linear FOV, right picture with a large FOV.

Figure 51: Gopro FOV linear VS large

5.2 Data Exportation

Once the trail is recorded, the user needs to transfer the .mp4 video from its GoPro to its computer. Then, import the footage in Kinovea software, set the reference distance and choose a point to track distance and speed frame per frame. A new .CSV file on the user's computer is created. This new file needs to be analysed by the dedicated python code that will perform data cleaning and processing to enable the user to visualise the data in the forms of graphs and the tuning suggestions inside an analyse report stored on the user's PC inside a .pdf file.

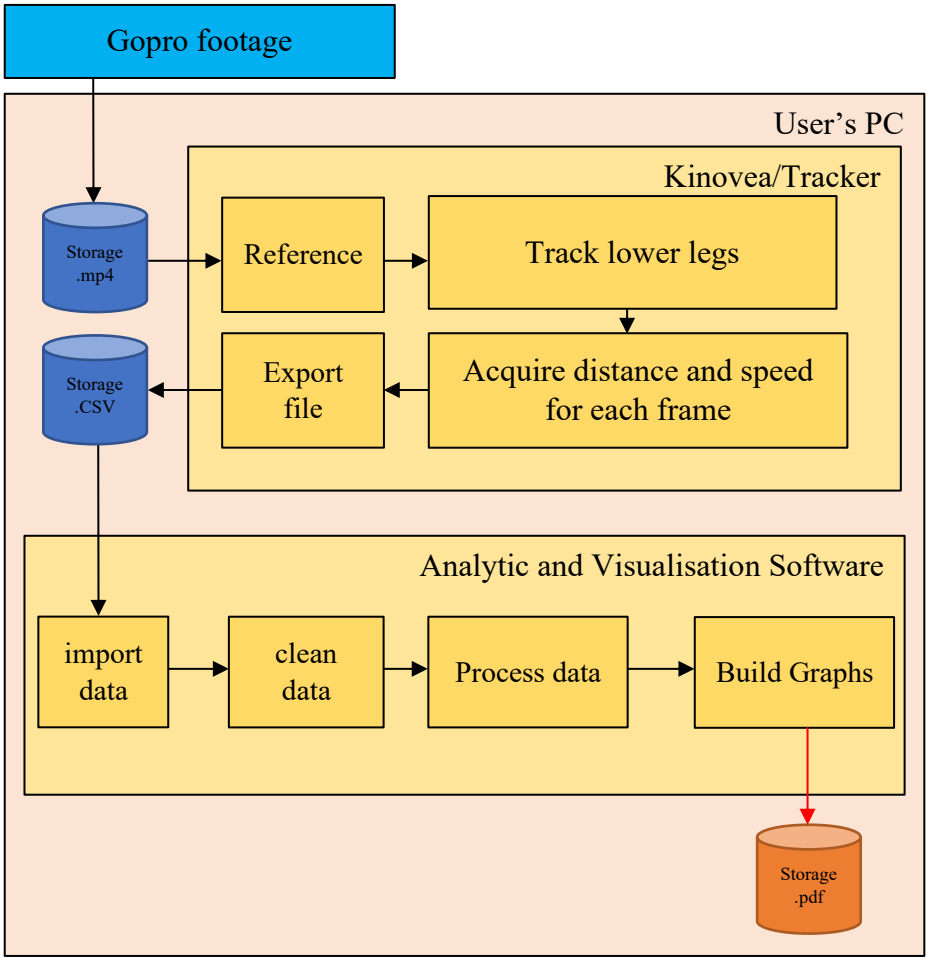


Figure 52: Gopro footage Software diagram

5.3 Tracker Versus Kinovea

The camera is mounted on the lower tube of the front triangle as close as possible to the frame. The inconvenience of this mount position appears when the handlebar is being steered. The fork moves inside the frame of the video. This is a problem for tracking software with a fixed coordinate system dependency such as Tracker.

If the camera were mounted on the fork this effect would vanish but in counterpart, the camera mount should stand back a lot to be able to frame all the stanchion needed for travel tracking, thus inducing significant vibration altering the travel tracking quality and problem while steering.

In contrast with Tracker, Kinovea proposes a coordinate system tracking nullifying this steering problem, since the coordinate system is dynamic. Consequently, Kinovea was chosen over Tracker as the motion analysis software.

On Kinovea, the origin of the coordinate system is chosen centred on the junction of the left stanchion and the crown, centre of the red grid in Figure 53.
Then, the calibration process can start; the user must select a frame of the video where the fork is fully extended. On the software, select the line tool and draw from the origin, down to the seal. This value is the reference for the calibration.

Figure 53: Kinovea coordinate system and calibration

Once the coordinate system and the calibration are done, the next step is to track the travel, in Kinovea terms, to track the horizontal position of a point. With the marker tool, place a point on the left seal. Choose the time interval to start and finish tracking, and the results can be displayed.

5.4 Kinovea results

On TT1 sometimes the tracking of the coordinate and or the tracking of the travel fails during quick steering change and lean angle. Thus, leading to unusable data.

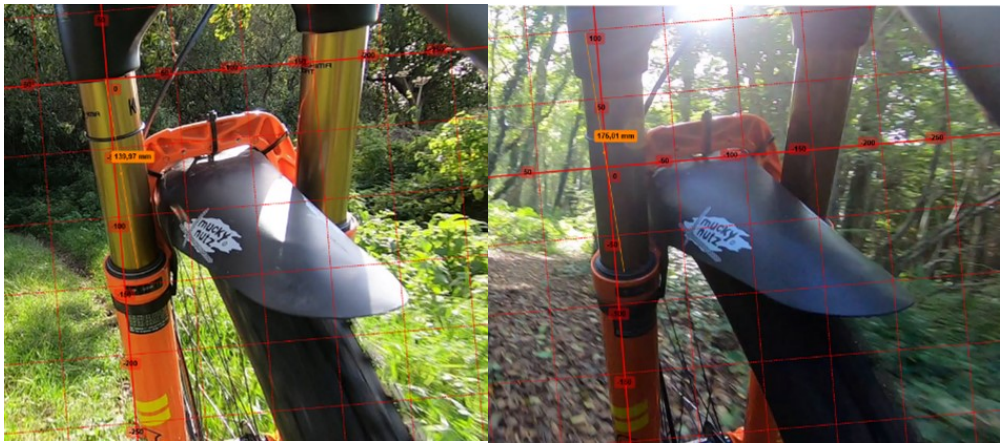


Figure 54: Kinovea tracking fails

The following graphs Figure 55 and 56 are extracted from the small drop test video [31].

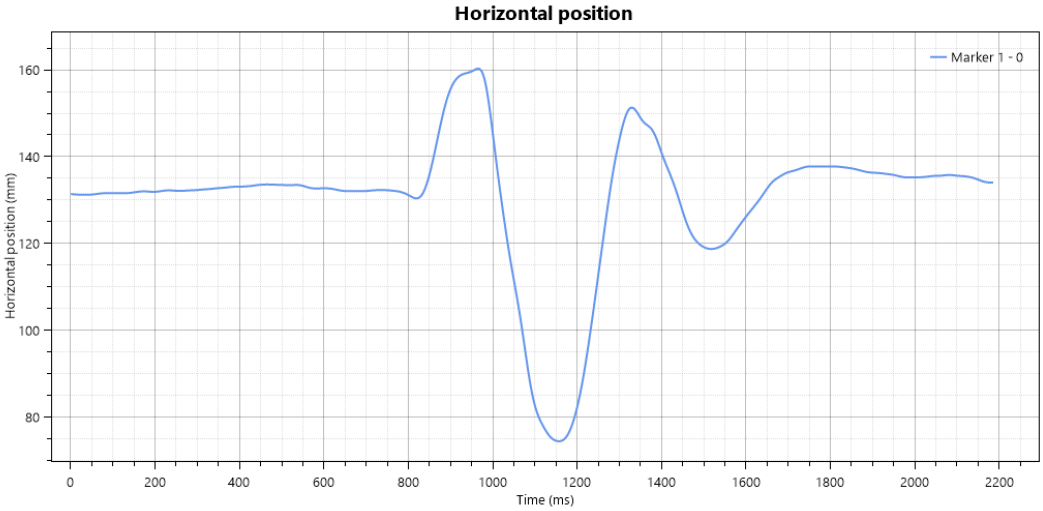


Figure 55: Small Drop Test, Horizontal position

The first 800ms are before the drop represents the SAG line (here around 12% SAG), the first peak shows the full extension of the fork (top value at 160mm), when the wheel is in the air, and so no forces activate it. Then, the wheel touches the ground, and the fork compresses (from 160mm to 75mm). The rebound performs its task, but here it creates two oscillations before returning to the SAG line, which is too much.

Besides position, Kinovea offers the possibility to compute the velocity. When the velocity hits 0m/s it means the travel stays constant. When the fork compresses the velocity increases negatively, alternatively a fork rebound results in a positive value.

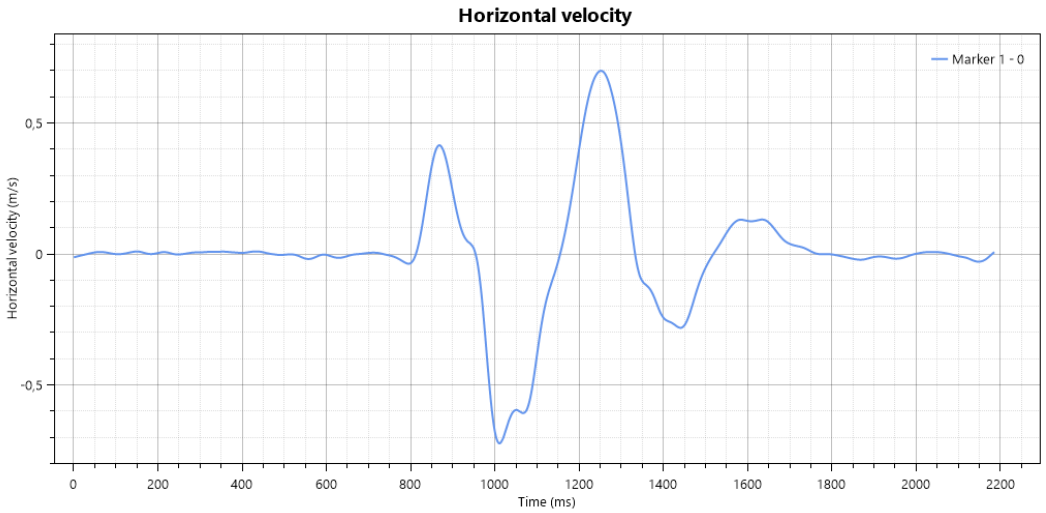


Figure 56: Small Drop Test, Horizontal velocity

Those graphs can be exported into a CSV file possessing two columns with headers. The first column represents the time in ms and the second the horizontal position or the horizontal velocity depending on the selected graph.

6. Analytic and Visualisation Software Design

This section is consecrated to the code used to analyse the Kinovea result; a consequent subsection is also dedicated to guidance by explaining with an illustrated method, how to achieve a good tuning.

6.1 Coding information

The code is written in Python 3.8, it inputs the two .CSV files from Kinovea and outputs a pdf displaying the necessary graph to understand the suspension comportment over the obstacle; an example of such pdf can be found here [32].

The core libraries used are:

- Panda, for extracting data from .CSV file and organising them.
- Matplotlib, for graph construction
- Fpdf, to output the results.

6.2 Graph construction

Two primary types of graphics must be achieved; the first type is based around horizontal displacement and the second type around horizontal velocity.

To begin with, NumPy oversees importing the corresponding files into the code; the files should be placed in a specific folder with a specific name because the program uses a relative path. After opening the files, the values are placed in data frames with relevant headers.
Secondly, the program verifies that all the values are in the 0-160mm range; if the case differs the corresponding extreme value replaces the value.
The velocity is then converted from m/s to mm/s, which is a better representation of the achieved speed values.
After this, matplotlib can be used to build the graphs. Information can be displayed on the graphs before it is exported in a single .pdf file created with the Fpdf library, besides, this .pdf file contains the date of its creation and the user's current tuning for better traceability of its tuning.

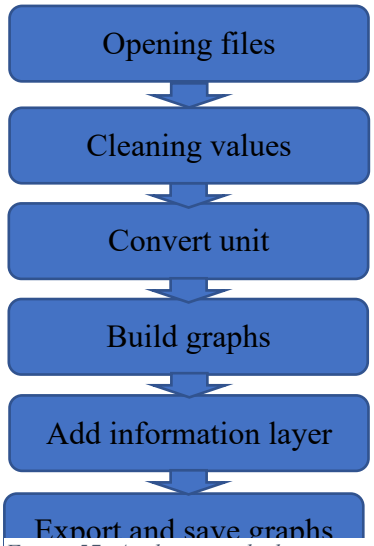


Figure 57: Analysing code diagram

6.2.1 Travel tracking

The primordial one is the travel tracking graph. It takes the position on the y-axis and times on the x-axis. A pattern in the T1SD output has been found. First, the value will mostly stay constant before forming a peak reaching top value. A line of code was dedicated to finding that peak. Then it computes the average value before it, to create an average line. This line can be used to state if the values reach back to where they were before the obstacle.

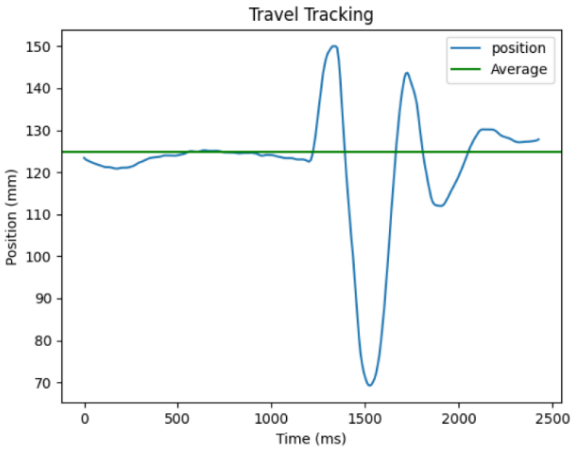


Figure 58: Travel tracking graph

6.2.2 Travel bar chart

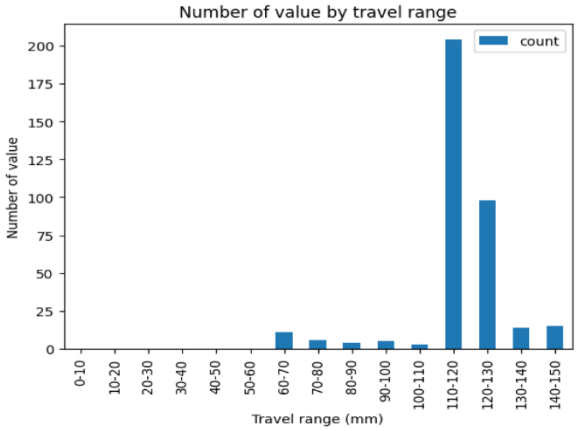


Figure 59: Travel bar chart

A subjacent bar chart is then created. For each 10mm a travel range is created, and the program counts how many values were in that range. It permits, for much longer tests, like trail test, to instantly notice where the fork mostly seats in its travel.

6.2.3 Velocity bar chart

The same process is performed with velocity, but this time, as opposed to fork travel, velocity is inconstant and will differ from each test. The ranges are subdivisions of the maximum velocity instead of being created with a fixed value. The user can change the number of those subdivisions by modifying one variable, by default the value is fixed to six, resulting in six ranges for compression velocity and six ranges for rebound velocity.

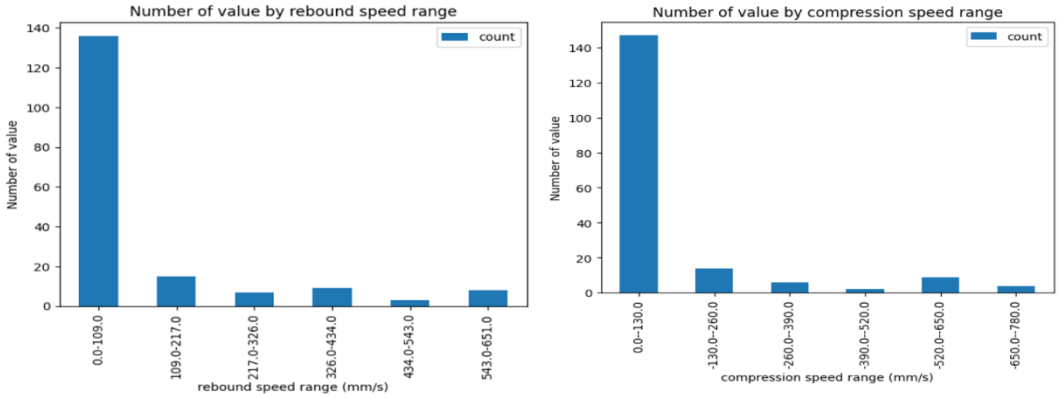


Figure 60: Velocity bar chart

6.2.4 Velocity graph

The last graph tracks the velocity of the compression and rebound; it has velocity values on the x-axis and time on the x-axis.
A background colour was applied, red for positive value (rebound) and blue for negative value (compression), following the Fox tuning knobs colour code.
Then an arbitrary distinction was made between high-speed and low-speed values at 157mm/s

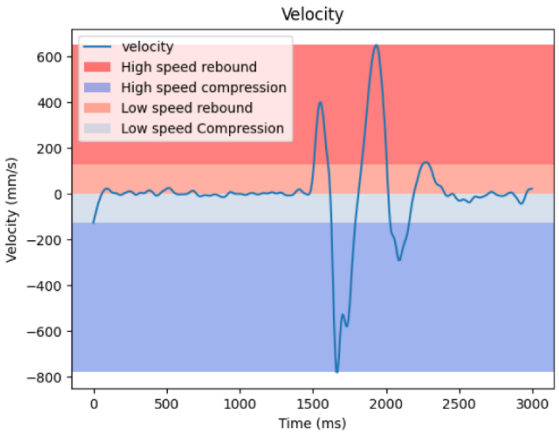


Figure 61: Velocity graph

6.3 Test method

First and foremost, the code should only be considered as a tool that permits to display graphs. To be able to tune the suspension correctly a method must be set.
A good starting point is Fox recommended settings, they can be found on their website and are dependent on the rider's weight for air spring pressure and rebound setting, but as we will prove they are too universal. I immediately changed the 20% SAG recommended to 27% that is more suited for an enduro fork.

A "Test 1 Small Drop" described in chapter 3.2 is performed and outputs the following travel diagram in Figure 62. The first 1000ms, before the obstacle, the travel remains constant as I am in a neutral position and nothing happens. Then the front wheel is in the air. The rebound expands the fork to its full travel inducing the peak at 1200ms. When the wheel hits the ground, the fork starts to compress, meaning the travel plummets to its lowest value at 1500ms. After the impact, the HSR tries to recover to the initial position (green line) but is set too fast, thus creating an overshoot (first oscillation). At the top of the overshoot, as we are unable to return to the original position, our body weight shifts to compress the fork. Since the HSR is too fast, the weight shifts are too abrupt and crosses again the initial position, thus, creating a smaller and second oscillation. To resume, a too consequent overshoot and as well as one unwanted oscillation.

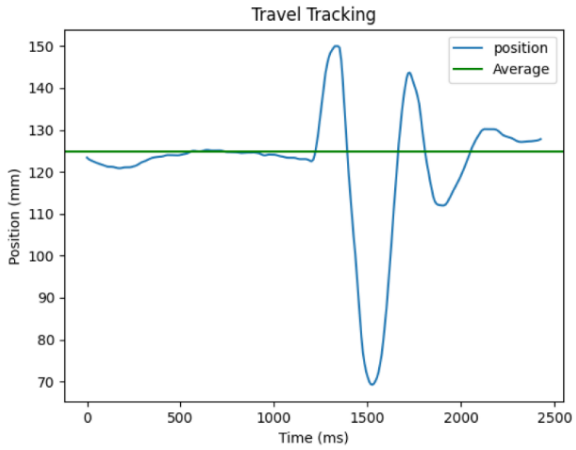


Figure 62: Fox recommended settings graph

The same test is performed with a slowest HSR and LSR to reduce the overshoot and the oscillations. After some adjustment I obtained the critical rebound graph Figure 63.

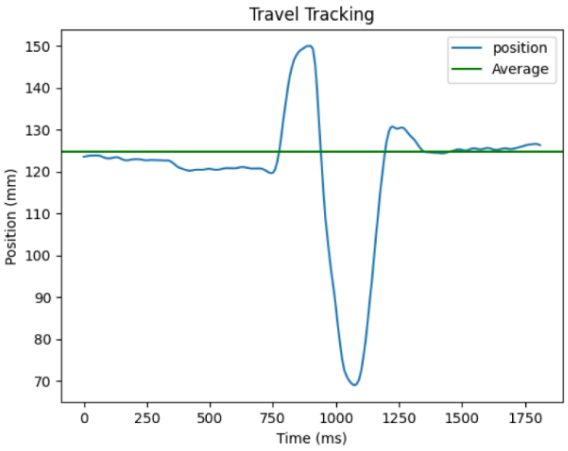


Figure 63: Critical rebound graph

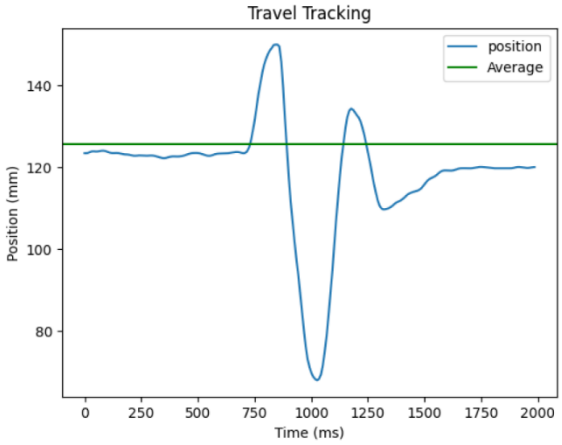


Figure 64: Low rebound graph

Below this critical setting, the rebound is too slow and unforceful to recover to the initial position. In Figure 64 the travel stabilises under the green line after the obstacle.

At first, one error I made during the test was to ride down the obstacle seated as it is easily repeatable in terms of body mass movement and position but after the impact, the fork was impotent to retrieve SAG value, since SAG is not obtained in a seated position but on a neutral position.

Once the critical LSR and HSR setting is obtained through this method, we want to test them in another obstacle configuration: stair, to test their reaction in rapid successive hits and adjust them, if necessary. This is the goal of T2SH.

To prevent the fork from bottoming-out we are using T4J. Depending on the size of the jump selected, the user should attest that the fork remains above the last 15% of the available travel. If there is too much travel consume increase the HSC. In the Figure 65 graph, the first compression peak refers to the body weight shift when the front wheel hits the change of angle from the jump's kick, the second compression peak corresponds to the jump landing and does not need more HSC clicks.

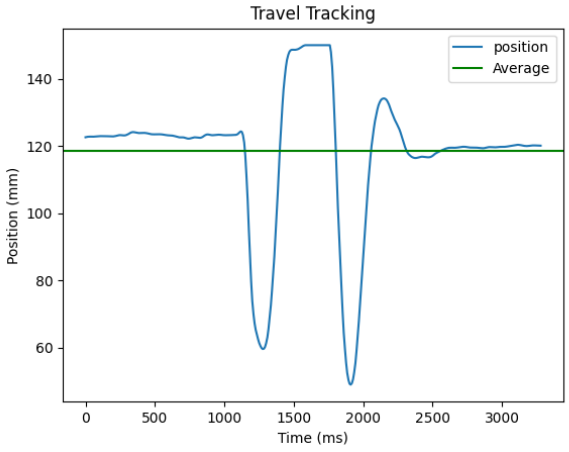


Figure 65: T4J graph

T3TC can be used for a discipline like Cross country where clearing technical climb is primordial, in Enduro this parameter is minor, and the test was ignored. The idea is to increase LSC for less body weight shift, which will help reduce pedal bob and clearing climbs but will highly sacrifice comfort and low-speed turn precision.

7. Results and Evaluation

7.1 Goals' evaluation

The goal was to create a cheap and accurate telemetry system (hardware + method) for the average rider. A multi-sensor was supposed to back-up the Gopro solution to give more credits to it, but it turns out that the complicated embedded Arduino board designed during this thesis was outclassed by the GoPro in terms of performance, reliability, and user-friendliness. The GoPro solution can carry alone the process of tuning a suspension, as shown by the following results.

A simple method has been put together for beginner riders to focus on finding an accurate SAG according to the rider's discipline and a critical rebound and average compression tuning. For more experienced riders, the different possible graphs results have been detailed and explained to enable the user to tune its suspension to its convenience.

Accustom average riders to suspension tuning means that this solution should work on most of the suspension. We have defined some rare incompatible suspension design and we decided to work only on the front suspension, but this solution could work with some rearrangements on a rear shock and even on those "incompatible" forks, but since they differ from what most of the riders possess, we separate them from our work.

The method was designed for a high-end fork with the most tuning (HSR, LSR, HSC, LSC) so this system can be interpreted for a fork with fewer adjustments.

7.2 Applicability of the work

This work revolves around the idea of possessing a Gopro to be applicable. As most riders already own a GoPro, this solution does not induce cost for them. In the other scenario, acquiring a Gopro is simple, the investment in a new one is as pricey as buying the Sram telemetry system, but Gopro is a well-distributed brand and can be easily found cheaper in the second-hand market. Furthermore, after the tuning process, the GoPro can be re-used as its designed role of action camera, whereas other telemetry systems become useless the moment after they have served their purpose.

The Gopro mount on the frame is detachable, it will keep the frame from deterioration and require only one piece of high-density foam, two zip ties and the included Gopro equipment.

7.3 Results

A relevant test was found to tune the rebound: T1SD. The compression settings were dialled to prevent bottom-out thanks to the T4J.

The GoPro method successfully performs rebound tuning with a 120Hz sampling rate.

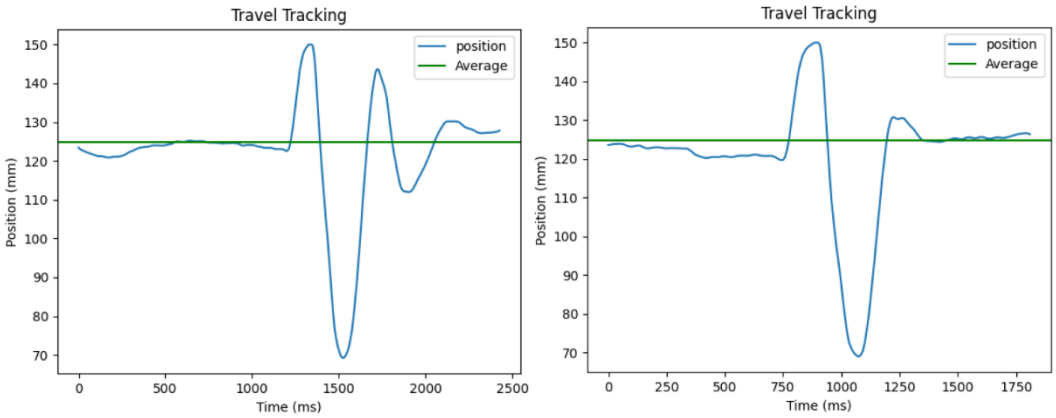


Figure 66: Before and after rebound tuning

The python program is flexible enough to handle any scenario of the tests. Few parts were hardcoded; all the results are delivered in a single PDF, including current tuning, date, and graphs.

The embedded electronics method was initially here to confirm or contrast the GoPro method, but it appears that it was less reliable due to the low sample rate, but mostly due to its acquiring systems. Besides, the system was insufficiently removable, non-waterproof, and dustproof. I was sceptical about the Gopro reliability in dark light, or the correctness of the video analysis software, but it resulted in a more reliable solution than the embedded electronics.

8. Conclusion

The main idea of the work was to determine a solution that could be suitable to tune a mountain bike front suspension and even achieve a trail-specific tuning. Since many riders admit to having difficulties with the tuning process and since a professional telemetry solution that could solve this problem is too costly (400-2000€) and will be useless once the tuning is done, I decided to use a GoPro camera that most of the riders already possess to prevent from inducing cost.

During this thesis work, I have learned certain things. First, in terms of electronics, I have learned that an accelerometer is inappropriate to be used for short and reliable displacement determination. Even though acceleration is the double integration of the distance, a random term is induced by the double integration that ruins the utilisation of the value. Another surprising fact I learned is that motion analysis software like Kinovea is more reliable than expected and in rare cases of tracking error the user can manually correct it. Obviously, with a thesis subject like this, I have gained meaningful knowledge in terms of suspension, but most importantly during the method creation process, I have learned how to isolate a specific tuning, how to set up a test and experiment on it to try to find cases where it could lead to errors, remedy the errors, and rectify the test.

Even though I have put all my efforts into this thesis work, it can still be improved. An immense improvement could be done on the practicality of the process, for example creating a connection between the GoPro and a smartphone to transfer the video of the test and crop it to have only the interesting parts. It could be beneficial if the motion analysis software could be ported into the phone, like that no need to have access to a computer to generate the .CSV file, this means that the suspension can be tuned in “real-time” along the trail instead of consecrated a day of ride to test different tuning and process them at home.
The second biggest improvement I could think of is to have an AI to interpret the outputted graph instead of the user. This could also lead to better analysis and maybe a trail-specific tuning instead of an average one.

9. Bibliography

[1] ‘FOX vs. RockShox: Mountain Bike Fork Guide’, *The Pro’s Closet*. [Online]. Available: https://www.theproscloset.com/blogs/news/fox-vs-rockshox-mountain-bike-fork-guide. [Accessed: 01-Jul-2020]

[2] ‘Fox 36 Tuning Guide 605-00-216-revA. [Online]. Available: https://www.ridefox.com/dl/bike/my21/605-00-216_RevA-36-Tuning-Guide.pdf. [Accessed: 01-Jul-2020]

[3] ‘StructureCycles Design’. [Online]. Available: https://structure.bike/pages/tech/#Design. [Accessed: 02-Jul-2020]

[4] ‘Lauf Grit’. [Online]. Available: https://www.laufcycling.com/product/lauf-grit. [Accessed: 02-Jul-2020]

[5] ‘RockShox RS-1 upside down fork — First look’, *BikeRadar*. [Online]. Available: https://www.bikeradar.com/news/rockshox-rs-1-upside-down-fork-first-look/. [Accessed: 02-Jul-2020]

[6] ‘String Encoders: How they work and what are their benefits’. [Online]. Available: https://www.pc-control.co.uk/string_encoders.htm. [Accessed: 15-Jul-2020]

[7] ‘What is a String Pot?’, *What is a string pot?* [Online]. Available: https://www.te.com/usa-en/products/sensors/position-sensors/potentiometric-sensors/cable-actuated-position-sensors/what-is-a-string-pot.html. [Accessed: 15-Jul-2020]

[8] ‘SussMyBike Home’, *SussMyBike*. [Online]. Available: https://www.sussmybike.com/. [Accessed: 01-Jul-2020]

[9] ‘Motion IQ technology’, *Motion Instruments*. [Online]. Available: https://motioninstruments.com/pages/technology. [Accessed: 01-Jul-2020]

[10] ‘What is a Linear Potentiometer?’ [Online]. Available: https://www.variohm.com/news-media/technical-blog-archive/what-is-a-linear-potentiometer-. [Accessed: 23-Jul-2020]

[11] ‘Motion Instruments IQ Datasheet Tracer Position Sensor’. [Online]. Available: https://cdn.shopify.com/s/files/1/0050/4997/4819/files/Tracer_Final_6.pdf?916. [Accessed: 01-Jul-2020]

[12] ‘Motion Instruments IQ Datasheet MIPS Position Sensor’. [Online]. Available: https://cdn.shopify.com/s/files/1/0050/4997/4819/files/MIPS_Final_6.pdf?916. [Accessed: 01-Jul-2020]

[13] ‘BYB Telemetry | World first universal MTB telemetry system’, *Bybshop*. [Online]. Available: https://www.bybtelemetry.com. [Accessed: 23-Jul-2020]

[14] ‘ShockWiz’. [Online]. Available: https://www.sram.com/en/quarq/series/shockwiz. [Accessed: 01-Jul-2020]

[15] K. Grund, ‘IoT Video Telemetry System Implemented on Mountain Bike’, Master Thesis, San José State University, 2017 [Online]. Available: https://static.kylegrund.com/Kyle+Grund+-+Thesis+Project+Report.pdf. [Accessed: 01-Jul-2020]

[16] STMicroelectronics, ‘World’s smallest Time-of-Flight ranging and gesture detection sensor’, p. 40, Apr. 2018.

[17] A. Aquib, ‘Ultrasonic Sensor and Arduino Tutorial’, *Medium*, 29-Feb-2020. [Online]. Available: https://medium.com/@aquibansari12377/ultrasonic-sensor-and-arduino-tutorial-89c38c81f103. [Accessed: 01-Sep-2020]

[18] ‘Andrextr - MTB Suspension’. [Online]. Available: https://www.andrextr.com/. [Accessed: 07-Sep-2020]
[19] ‘Tracker Video Analysis and Modeling Tool for Physics Education’. [Online]. Available: https://physlets.org/tracker/. [Accessed: 15-Jul-2020]
[20] *FREE Suspension Data Acquisition (MTB Rear suspension Ep.6)*. [Online]. Available: https://www.youtube.com/watch?v=4BRbtkiW2tg. [Accessed: 07-Sep-2020]
[21] ‘Kinovea - Terms of use’. [Online]. Available: https://www.kinovea.org/help/en/003.html. [Accessed: 29-Sep-2020]
[22] ‘COMMENCAL 2021 | CADRE META HT AM SAND’. [Online]. Available: https://www.commencal-store.com/cadre-meta-ht-am-sand-c2x31374491. [Accessed: 17-Aug-2020]
[23] *How to CORRECTLY setup rebound damping (MTB rear suspension Ep.5)*. 2016 [Online]. Available: https://www.youtube.com/watch?v=BiHQd4mzl3Y. [Accessed: 15-Jul-2020]
[24] ‘Gopro, extracting the metadata in a useful format’, 10-Apr-2017. [Online]. Available: https://community.gopro.com/t5/GoPro-Telemetry-GPS-overlays/Extracting-the-metadata-in-a-useful-format/gpm-p/40293#M1. [Accessed: 01-Jul-2020]
[25] A. Industries, ‘Adafruit HC-SR04 Ultrasonic Sonar Distance Sensor + 2 x 10K resistors’. [Online]. Available: https://www.adafruit.com/product/3942. [Accessed: 02-Jul-2020]
[26] A. Industries, ‘Adafruit VL53L0X Time of Flight Distance Sensor - ~30 to 1000mm’. [Online]. Available: https://www.adafruit.com/product/3317. [Accessed: 02-Jul-2020]
[27] A. Industries, ‘Adafruit LSM6DSOX 6 DoF Accelerometer and Gyroscope’. [Online]. Available: https://www.adafruit.com/product/4438. [Accessed: 02-Jul-2020]
[28] ‘Bluetooth Terminal – Applications sur Google Play’. [Online]. Available: https://play.google.com/store/apps/details?id=Qwerty.BluetoothTerminal&hl=fr. [Accessed: 07-Sep-2020]
[29] ‘double - Arduino Reference’. [Online]. Available: https://www.arduino.cc/reference/en/language/variables/data-types/double/. [Accessed: 19-Aug-2020]
[30] ‘MIT App Inventor | Explore MIT App Inventor’. [Online]. Available: https://appinventor.mit.edu/. [Accessed: 07-Sep-2020]
[31] *Test 1 Small Drop (week 42, 2020)*. 2020 [Online]. Available: https://www.youtube.com/watch?v=ZfvMHSKvCzg&feature=youtu.be&fbclid=IwAR146RK_0SrONEH6Cc_BxTc9D9iIVE7BAkvsv3YVFFoX5txahLwrPGSlXOE. [Accessed: 20-Oct-2020]
[32] JeremyLesauvage, *JeremyLesauvage/Thesis-A-Multi-Sensor-Approach-to-Optimise-Mountain-Bike-Suspension*. 2021 [Online]. Available: https://github.com/JeremyLesauvage/Thesis-A-Multi-Sensor-Approach-to-Optimise-Mountain-Bike-Suspension. [Accessed: 23-Feb-2021]

10. Appendices

Analyse python code available at [32]:

HTTPS://GITHUB.COM/JEREMYLESAUVAGE/THESIS-A-MULTI-SENSOR-APPROACH-TO-OPTIMISE-MOUNTAIN-BIKE-SUSPENSION

```python
from tkinter.filedialog import askopenfilename

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import datetime
import os
from PIL import Image, ImageDraw, ImageFont
from fpdf import FPDF

"""
 This script is only to analyse the Kinovea results from test 1 small drop, the file to analyse need to be in the
 folder T1SD_result with the name T1SD_P.csv for the position and T1SD_V.csv for the velocity
"""

stanchions_length = 160
fork_travel = 150

# Open file and create Pandas DataFrame while changing the header, specifying the separator and defining data type,
# we also need to change the decimal delimiter from , to . to convert str value into float
df_p = pd.read_csv('T1SD_result/T1SD_P.csv', sep=';', names=['time', 'position'], header=0,
                   decimal=',', dtype={'time': int, 'position': float})

df_v = pd.read_csv('T1SD_result/T1SD_V.csv', sep=';', names=['time', 'velocity'], header=0,
                   decimal=',', dtype={'time': int, 'velocity': float})

# CLEANING VALUES
# replacing position value greater than the stanchion length (impossible value) by the maximum value (stanchion_length)
df_p.loc[(df_p.position > stanchions_length), "position"] = stanchions_length
# offset the value by stanchion length - fork travel
df_p.loc[:, "position"] -= stanchions_length - fork_travel
# replacing position value below 0 by 0
df_p.loc[(df_p.position < 0), "position"] = 0
# Convert velocity unit to mm/s (1 mm/s = 0.001 m/s)
df_v.loc[:, "velocity"] *= 1000

# PRINT DATAFRAMES FOR VERIFICATION
# print(df_p)  # show the position DataFrame
# print(df_v)  # show the velocity DataFrame

# CREATE TRAVEL TRACKING GRAPH
df_p.plot(x="time", y="position", kind='line', title='Travel Tracking', xlabel='Time (ms)', ylabel='Position (mm)')
# determine the first peak value corresponding to the fork full extension before the wheel hit the ground
first_peak = df_p.loc[(df_p.position == df_p.position.max()), 'time'].tolist()[0]
# compute the average value between tracking start and the peak value
average_before_peak = df_p.loc[(df_p.time <= first_peak), "position"].mean()
# draw this average line corresponding to "sag" after compression settings
plt.axhline(y=average_before_peak, color='green', linestyle='-', label='Average')
plt.legend()
plt.savefig('T1SD_result/Travel_Tracking.png') # save the graph
plt.show()

# CREATE VELOCITY GRAPH
df_v.plot(x="time", y="velocity", kind='line', title='Velocity', xlabel='Time (ms)', ylabel='Velocity (mm/s)')

# Low speed / high speed range for graph building and analysis
hsr_limit = df_v.velocity.max()  # check the fastest rebound speed
lsr_limit = 127  # exact number to define
lsc_limit = -127  # exact number to define
hsc_limit = df_v.velocity.min()  # check the fastest compression speed

# If the value are inferior do not show high velocity facecolor
# adjust the range of low velocity to the fastest compression speed and rebound speed
if df_v.velocity.max() > lsr_limit:
    plt.axhspan(lsr_limit, hsr_limit, facecolor='red', alpha=0.5, label="High speed rebound")

if df_v.velocity.min() < lsc_limit:
    plt.axhspan(lsc_limit, hsc_limit, facecolor='royalblue', alpha=0.5, label="High speed compression")

plt.axhspan(0, lsr_limit, facecolor='tomato', alpha=0.5, label="Low speed rebound")
plt.axhspan(0, lsc_limit, facecolor='lightsteelblue', alpha=0.5, label="Low speed Compression")

plt.legend()
plt.savefig('T1SD_result/velocity.png')
plt.show()

# CREATE BAR CHART FOR NUMBER OF VALUE WITHIN TRAVEL RANGE
# Define number of value by travel range and create a DataFrame
array_travel_range = np.arange(0, fork_travel, 10)
count_travel_range = []
range_travel = []
for x in array_travel_range:
    count_travel_range.append(df_p.loc[(df_p.position <= x + 10) & (df_p.position > x), "position"].count())
    range_travel.append(str(x) + '-' + str(x + 10))

df_count_travel_range = {'count': count_travel_range, 'range': range_travel}
df_travel_range = pd.DataFrame(df_count_travel_range)  # DataFrame construction
df_travel_range.plot(x="range", y="count", title='Number of value by travel range',
                     kind='bar', xlabel='Travel range (mm)', ylabel='Number of value')
```

```
# save the graph, with bbox_inches='tight' since the graph exceed the security limit in height it will be crop otherwise
plt.savefig('T1SD_result/count_travel_range.png', bbox_inches='tight')
plt.show()



# REBOUND SPEED BAR CHART
n_bar = 6  # The bar charts for rebound and comprssion has 6 ranges calculated from max value
range_delimiter_rebound = hsr_limit / n_bar  # define the value to increment range for rebound speed
range_delimiter_compression = hsc_limit / n_bar  # define the value to increment range for compression speed
array_rspeed_range = np.arange(0, hsr_limit, range_delimiter_rebound)
count_rspeed_range = []
range_rspeed = []
for y in array_rspeed_range:
    count_rspeed_range.append(
        df_v.loc[(df_v.velocity <= y + range_delimiter_rebound) & (df_v.velocity > y), "velocity"].count())
    range_rspeed.append(str(round(y)) + '-' + str(round(y + range_delimiter_rebound)))

df_count_rspeed_range = {'count': count_rspeed_range, 'range': range_rspeed}
df_rspeed_range = pd.DataFrame(df_count_rspeed_range)  # DataFrame construction
df_rspeed_range.plot(x="range", y="count", title='Number of value by rebound speed range',
                     kind='bar', xlabel='rebound speed range (mm/s)', ylabel='Number of value')
plt.savefig('T1SD_result/count_rspeed_range.png', bbox_inches='tight')
plt.show()

# COMPRESSION SPEED BAR CHART
array_cspeed_range = np.arange(0, hsc_limit, range_delimiter_compression)
count_cspeed_range = []
range_cspeed = []
for z in array_cspeed_range:
    count_cspeed_range.append(
        df_v.loc[(df_v.velocity >= z + range_delimiter_compression) & (df_v.velocity < z), "velocity"].count())
    range_cspeed.append(str(round(z)) + '-' + str(round(z + range_delimiter_compression)))

df_count_cspeed_range = {'count': count_cspeed_range, 'range': range_cspeed}
df_cspeed_range = pd.DataFrame(df_count_cspeed_range)  # DataFrame construction
df_cspeed_range.plot(x="range", y="count", title='Number of value by compression speed range',
                     kind='bar', xlabel='compression speed range (mm/s)', ylabel='Number of value')
plt.savefig('T1SD_result/count_cspeed_range.png', bbox_inches='tight')
plt.show()

# ASK THE USER FOR ITS SETTINGS DURING THIS TEST
sag_percent = input("Enter amount of sag in %: ")
hsc_user = input("Enter amount of HSC click from firmest to lightest : ")
lsc_user = input("Enter amount of LSC click from firmest to lightest : ")
hsr_user = input("Enter amount of HSR click from fastest to slowest : ")
lsr_user = input("Enter amount of LSR click from fastest to slowest : ")
token_user = input("Enter amount of token : ")
img = Image.new('RGB', (425, 70), color=(70, 70, 70))
font = ImageFont.truetype("consola.ttf", 18, encoding="unic")
canvas = ImageDraw.Draw(img)
canvas.text((10, 10), "HSC: " + hsc_user + "/16", fill='#FFFFFF', font=font)
canvas.text((10, 40), "LSC: " + lsc_user + "/14", fill='#FFFFFF', font=font)
canvas.text((135, 10), "HSR: " + hsr_user + "/8", fill='#FFFFFF', font=font)
canvas.text((135, 40), "LSR: " + lsr_user + "/16", fill='#FFFFFF', font=font)
canvas.text((250, 10), "SAG: " + sag_percent + "%", fill='#FFFFFF', font=font)
canvas.text((250, 40), "TOKEN: " + token_user, fill='#FFFFFF', font=font)
img.save('T1SD_result/current_settings.png')
img.show()

# CREATE THE PDF WITH ALL THE RESULT
pdf_title = input("Enter the subtitle of the PDF : ")
pdf = FPDF()
pdf.add_page()
pdf.set_font('Arial', 'B', 16)
pdf.cell(0, 10, 'Test Result', ln=1, align='C')
pdf.set_font('Arial', '', 12)
pdf.cell(0, 10, pdf_title, ln=1, align='C')
today_date = datetime.datetime.now()
pdf.cell(0, 10, txt=str(today_date), ln=1, align='C')
# 1px= 0.264583mm
pdf.image('T1SD_result/Travel_Tracking.png', x=None, y=None, w=float(640 * 0.264583 * 0.7),
          h=float(480 * 0.264583 * 0.7))
pdf.image('T1SD_result/count_travel_range.png', x=None, y=None, w=float(640 * 0.264583 * 0.7),
          h=float(480 * 0.264583 * 0.7))
pdf.image('T1SD_result/Velocity.png', x=None, y=None, w=float(640 * 0.264583 * 0.7), h=float(480 * 0.264583 * 0.7))
pdf.image('T1SD_result/count_rspeed_range.png', x=None, y=None, w=float(640 * 0.264583 * 0.7),
          h=float(480 * 0.264583 * 0.7))
pdf.image('T1SD_result/count_cspeed_range.png', x=None, y=None, w=float(640 * 0.264583 * 0.7),
          h=float(480 * 0.264583 * 0.7))
pdf.cell(0, 10, txt="Settings for this test", ln=1)
pdf.image('T1SD_result/current_settings.png', x=None, y=None, w=float(425 * 0.264583), h=float(70 * 0.264583))
pdf.output('T1SD_result/'+pdf_title+'.pdf', 'F')

# DELETING INTERMEDIATE FILES
os.remove("T1SD_result/Travel_Tracking.png")
os.remove("T1SD_result/count_travel_range.png")
os.remove("T1SD_result/Velocity.png")
os.remove("T1SD_result/count_rspeed_range.png")
os.remove("T1SD_result/count_cspeed_range.png")
os.remove("T1SD_result/current_settings.png")
```