Luca Floreau

Student # 12-134269

Bachelor Games Programming

How can we improve the position estimation of a smartphone using inertial sensors during fast movements?

BACHELOR THESIS – MODULE CMN 6302.1

Farhan Elias, Siorak Nicolas

August 31, 2021

SAE INSTITUTE - GENEVA

Words : 12110

Summary

Motion-based games provide a unique experience for their players. It greatly increases the immersion of the player in the game. Unfortunately, the infrastructures needed for these games are often expensive. In parallel, smartphones, composed of a lot of useful technologies, are democratized more and more.
Therefore, the goal is to research a way to develop motion-based games through smartphones. Currently, there are two main technologies to track the position of a smartphone : optical tracking and inertial tracking. This research focuses on the second point and will answer the question : How can we improve the position estimation of a smartphone using inertial sensors during fast movements ?
The research begins with a study of what is currently being done in position estimation. This part aims to present multiple algorithms and their strengths and weaknesses. It's also supported by an analysis of three similar projects.
On this basis, three solutions are presented and the criteria for proper functioning are chosen. This leads to a list of tests that evaluate the effectiveness of the algorithms. For this work, the choice was to prioritize accuracy over universality, which means that the goal is to obtain accurate position estimation for specific movements.
After the analysis of the results, it has been determined that the efficiency of an algorithm depends mainly on the movement. Indeed, the parameters of the algorithm must be adapted to the user's movements. However, each algorithm can only be used for only specific movements. Further research needs to be done to increase the universality of the algorithm enough to be used by the players.

Keywords : position estimation, inertial sensors, dead reckoning, inertial navigation, 3D localization, Unity, motion-based game, smartphone

Preface

This work was carried out as part of a game programming bachelor at the SAE Institute in Geneva. It was done with the objectives of demonstrating and applying the skills acquired during the training to create a product that can be used by others.
In my personal experience, motion-based games provide a unique experience for their players. It greatly increases the immersion of the player in the game. Unfortunately, the infrastructures needed for these games are often expensive. I chose this particular topic in order to research an alternative to make motion-based games more accessible.

Acknowledgements

This work would never have been done without the help of these people :
- Elias Farhan, my teacher who made sure to keep the project on the right track and helped me shape it
- Nicolas Siorak, for his help structuring the whole article
- Camille Tamone, for her help on the wording and writing

As the author, Luca Floreau, I testify to having personally done and written this thesis.
I also certify that I have not resorted to plagiarism and have conscientiously and clearly mentioned all borrowing from others.

Table of contents

Introduction

Context

Since the release of the iPhone in 2007, smartphones have become more and more popular. In 2020, more than 67% of the world population used a smartphone and more than 75% of the population in Europe and North America (GSM Association, 2021).

Smartphones are full of sensors allowing them to detect gravity, magnetism, infrared, brightness, rotation or acceleration. All of these sensors allow smartphones to be used in many ways (Nield, 2020).

On the other hand, motion-based games are also becoming more and more popular. Indeed, it is one of the parties of Virtual Reality (VR), the popularity of the latter is on the rise (Statista, 2021). However, these consoles are still really expensive. Therefore, is it possible to use a smartphone for motion-based games?

Definition

To use a smartphone as a motion-based controller, it is important to track its position correctly. More precisely, it is important to detect the 6 Directions of Freedom (6-DoF), that is the monitoring of the 3 axes of position and 3 axes of rotation of the controller. For this, several methods are used : optical tracking which is based on a camera to define the position of an object, acoustic tracking which is based on sound, magnetic tracking which is based on magnetic fields and inertial tracking.

Inertial tracking is the ability to predict the position and orientation of a moving object without the need for external resources. It generally uses an accelerometer (tool recording the accelerations) and a gyroscope (tool recording the orientation). It will then use dead-reckoning algorithms to predict a position from rotation and acceleration.

Research question and objectives

Regardings those previous points, it is interesting to ask the question "How can we improve the position estimation of a smartphone using inertial sensors during fast movements ?".

Therefore, the objective of this research is to develop an algorithm allowing the tracking of the position of a smartphone during a short but fast movement. This algorithm must be able to estimate a recognizable movement and detect a static position.

Plan

This work is divided into five parts. The first part is a general state of inertial motion tracking in games. It first presents motion tracking in industry and video games. And then it presents the inertial tracking and which algorithm has been developed to improve the estimation.
The second part is an analysis of three projects using inertial tracking. These analyses allow understanding in detail the choice of the algorithm.
The following part explains the environment of the tests and describes the different tests and parameters of these tests.
The fourth part is a description of the implementation of the algorithm developed and a description of the parameters applied to these algorithms. It then presents the results of the different algorithms for each test.
The last part is an analysis of the result describing how the results reach the goals and which parameters influence the accuracy of the result.

1 Motion Tracking in Video Games

This chapter presents motion tracking and its uses, more particularly in video games. It then describes the tools, their function, and problems of inertial position tracking. Finally, it will discuss the techniques developed to overcome these problems.

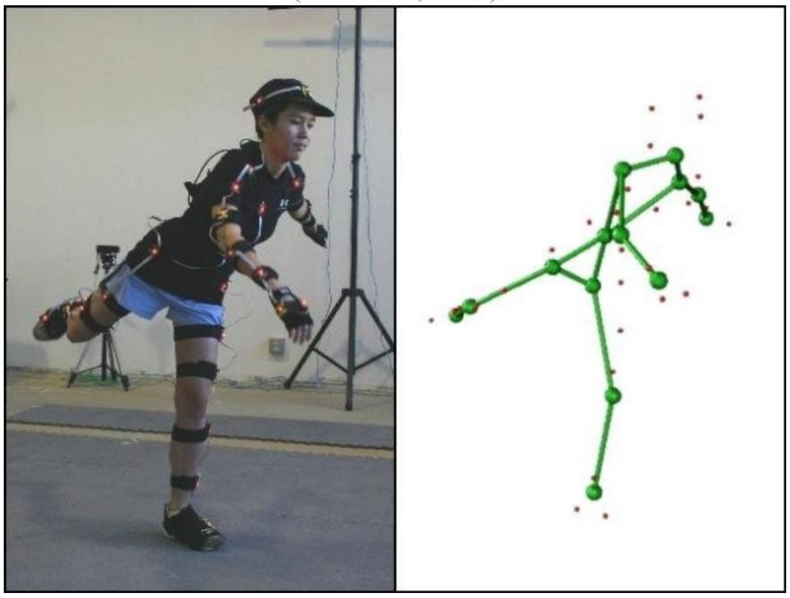
1.1 Usage of motion Tracking

1.1.1 Different types of motion tracking

Motion tracking is the ability to detect the movement of a device in a space. There is a multitude of technologies developed in order to perform motion tracking. The most common technology is the Global Positioning System (GPS). The GPS is a service using data from many satellites broadcast to a receiver which computes latitude, longitude and altitude thanks to the time duration between the signal reception and the broadcast time(*Satellite Navigation - GPS - How It Works*, 2021). A variety of industries are using them for navigation tracking. It is a good solution to track positions in big distances but it has 7.8 meters of inaccuracy and it is less efficient in indoor environments which impede the motion tracking of body movements.

For motion body tracking, the most commonly used technology is optical tracking. Optical tracking is a system that uses visual information in order to track the user's motion. It can be done in multiple ways. The most frequent way is the use of an external camera tracking the user device in order to estimate its position in the space. It can be done with one or more cameras which can be video or infrared(*Optical Tracking - an Overview | ScienceDirect Topics*, n.d.).
This system is used in various industries such as animation and visual effects to capture body movements.

Figure 1.1.1.B : Optical motion tracking system. Subject during the capture session wearing a suit with infrared markers (left). 3D human pose compute from the observed position of each marker (right) (Kirk et al., 2005)



Another way to do optical tracking is to use a camera on the device estimating its own position from a tracker around it. This tracker can be infrared or some environmental feature. This technique is particularly used for augmented reality, tracking environmental features to position the device inside the environment and overlay virtual images on the reality.

It is the most popular system for body tracking because it is reliable and accurate. However, these systems need exterior constraints such as external cameras, trackers or a reliable environment.

Another technology for motion tracking is inertial tracking. Inertial tracking is the use of measurement from an Inertial Measurement Unit (IMU) including at least an accelerometer in order to estimate the orientation, velocity and position of a device (Zanetti & D’Souza, 2020).

This system is used in many industries for navigation such as planes, rockets or spacecraft. It is a good way to determine position only with information inside the device.

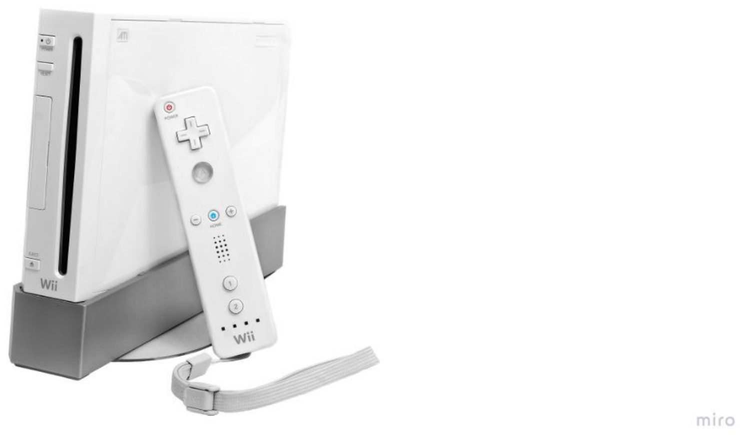
Then there is a wide range of systems of motion tracking such as acoustic tracking using echolocation, magnetic tracking using a magnetic field (‘VR Positional Tracking’, 2021), wifi tracking using wifi hotspot or network tracking using the service provider's network infrastructure (‘Indoor Positioning System’, 2021).

Finally, the most widely used way for motion tracking is hybrid-system tracking. It uses the combination of multiple technologies in order to estimate the position of the device. There is a multiplicity of technology in mobile tracking using a hybrid between GPS, Bluetooth and wifi tracking (*AlterGeo - Global Location Technology Provider*, n.d.) or in sport using a combination of optical and inertial tracking to analyze an athlete motion (Lin et al., 2018).

1.1.2 Motion Tracking in video games

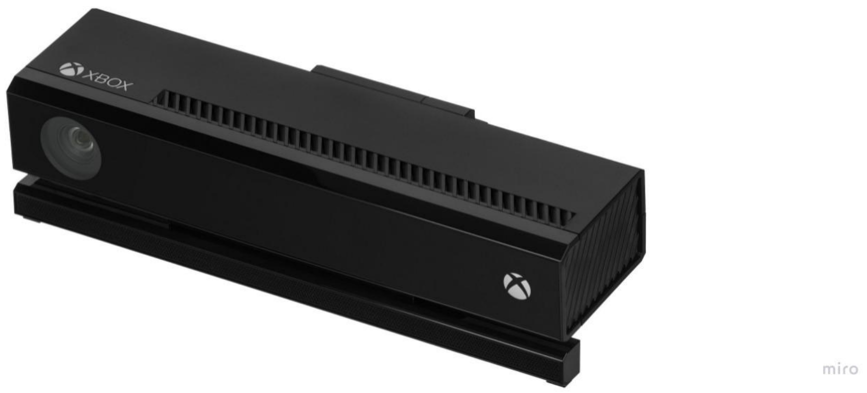
Motion Tracking allows the player to be more immersed in a video game. The console which popularized motion tracking in video games is probably the Wii. The Wii is a Nintendo console using a Wiimote to track the motion of the player ('Wii', 2021). It uses the player's motion for various games such as sport or dancing games. The Wiimote uses a hybrid of optical and inertial motion. Indeed, the Wiimote contains an accelerometer to detect acceleration and orientation and a camera tracking infrared LED next to the console.

Figure 1.1.2.A : The Wii console by Nintendo featured with the Wiimote (Wikipedia, 2010)



Another use of body tracking in video games is the Kinect. The Kinect is an Xbox extension composed of one camera and one depth camera recognizing a player and tracking their movement. It is also used for sport, dance or arcade games ('Kinect', 2021).

Figure 1.1.2.B : The Xbox One's Kinect, (Wikipedia, 2014)



Finally, another popular use of motion tracking in video games is in VR. It is a technology that simulates the physical presence of a player in a virtual environment. For this, the VR generally uses a headset tracking the position and orientation of the player head while providing a stereoscopic head-mounted display and two motion controllers. This technology generally uses a hybrid system combining optical tracking from external cameras and inertial tracking from intern acceleration, rotation and magnetic sensors ('Virtual Reality', 2021).

Figure 1.1.2.C : Oculus Quest 2 (Oculus, 2020)

In this paper, we will focus on how inertial tracking works and how we can use it for body motion tracking with a smartphone.

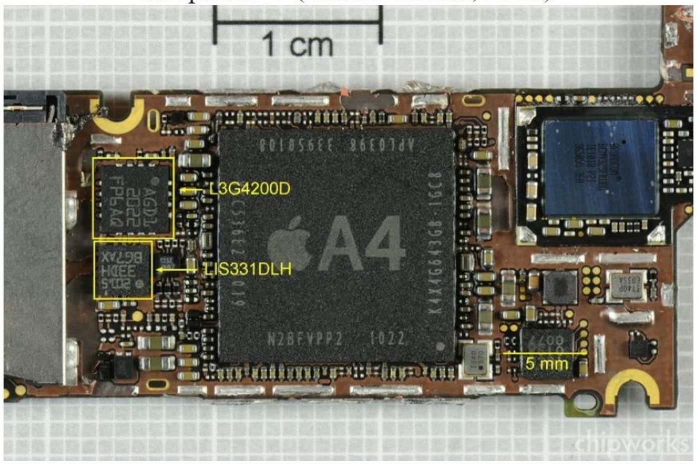
1.2 Technology and problems of inertial tracking

Inertial tracking is based on the data received from inertial sensors.

1.2.1 Technology of inertial sensor

Inside modern smartphones, there are many sensors in order to increase accessibility such as orientation adaptation or pedometers (EI, 2019). In general, the sensors are combined inside an IMU. These IMUs are usually composed of a 3-axis accelerometer and a 3-axis gyroscope sensor.

Figure 1.2.1.A : STMicroelectronics LIS331DLH accelerometer and the L3G4200D gyroscope inside an Iphone 4 (Dixon-Warren, 2010)

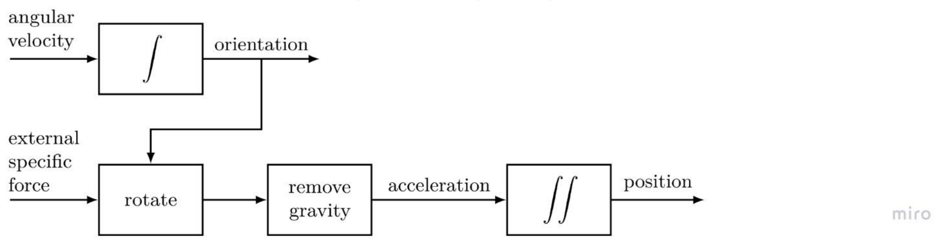


A 3-axis accelerometer is a device composed of 3 accelerometers that measure the acceleration of the movement on 3 different axes. An accelerometer is composed of a central object with enough mass to move depending on the acceleration and a sensor on the side to detect the movement of this mass. To detect the motion, the accelerometer can use capacitors, piezoelectric crystals or electrodes as in MEMS accelerometers (*How Accelerometers Work | Types of Accelerometers*, 2009). MEMS stands for Micro Electro Mechanical System. These are micrometre sensors using electrodes for mechanical sensoring. (EI, 2019). These are also compounds of the MEMS gyroscope.

A 3-axis gyroscope (also called gyrometer) is a device composed of 3 gyroscopes that measure the angular speed of the system on 3 different axes. There are several designs for the gyroscope such as the tuning fork, the vibrating-wheel, the wine glass resonator or the Foucault pendulum (Bernstein, 2003). In this work, only the tuning fork will be discussed because it's the one used on smartphones. The Tuning Fork Gyroscope uses the Coriolis Effect. This effect detects a fictitious force on a moving mass when the frame of reference is rotating. This allows the gyroscope to detect how the frame of reference is rotating. The MEMS gyroscopes use a pair of masses oscillating, allowing the detection of the Coriolis effect on electrodes.(*Gyroscopes - MEMS and Sensors - STMicroelectronics*, n.d.)(Bernstein, 2003)(Google TechTalks, 2010)

1.2.2 Using these data

Figure 1.2.2.A : Schematic illustration of dead-reckoning, where the accelerometer measurements (external specific force) and the gyroscope measurements (angular velocity) are integrated to position and orientation. (Kok et al., 2017)



To calculate a translation of a device using only the acceleration and the orientation, the dead reckoning is used.
The dead reckoning is “the process of calculating one's position, especially at sea, by estimating the direction and distance travelled rather than by using landmarks or astronomical observations.”(‘Dead Reckoning’, 2021). To perform dead reckoning from the gyroscope and the accelerometer, the first step is to obtain the orientation of the device.
Raw gyroscope values return the angular speed, so, to obtain the attitude of the device, the first step is to integrate the angular speed.
Raw accelerometer values are the local acceleration plus the gravity. Thanks to the calibration, it is possible to calculate the gravity and the local acceleration of the device.
The second step of dead reckoning is to obtain the acceleration on the global space.
By transforming the orientation into a rotation matrix, it is possible to rotate the local acceleration into a global acceleration.
In theory, to obtain the position, it only needs to integrate the acceleration two times.
By integrating angular speed into orientation and acceleration into position, it introduces integration drift. Indeed, due to the double integration, a small inaccurate value on the acceleration will grow quadratically with time. Furthermore, a small inaccuracy in the estimated orientation can cause a big difference of position on a long distance. That is why algorithms have been developed in order to minimize inaccurate values and reduce drift.

1.3 Algorithm to improve position estimation

1.3.1 Calibration

The first step to reduce data inaccuracy is calibration. The raw data produced by the accelerometer must be processed before use. The first step is to remove the initial bias of the device. A bias describes a repeatable error resulting in a total difference from the true value. It can, for example, be due to the temperature.
The second step is to detect an error due to the scaling of the values. The scaling is a factor that amplifies or reduces the measured value to obtain the real value.
In our case, a standard scaling is the conversion of the unit measure. The accelerometer provides a measure in gravitational acceleration (g) and it is generally converted to metre per second squared (m/s^2).
The final calibration step is the removal of noise. The accelerometer and the gyroscope are constantly corrupted by noise. "Noise is an unwanted signal output that does not represent true vibration." (*Understanding the Accelerometer Noise Specification | Wilcoxon Sensing Technologies*, n.d.). They are produced because the accelerometer is an electronic amplifier. In our case, we are looking to establish the acceleration given by measurement acceleration. Therefore, the noise can be expressed using Conditional Probability Distribution. The accelerometer noise can be estimated with additive white Gaussian noise. Indeed, when the noise is Gaussian, it can be described using its mean and its covariance (Kok et al., 2017). The covariance is the correlation between two values.
A popular way to remove Gaussian noise from a data set is the use of smoothing algorithms.

1.3.2 Smoothing Algorithms

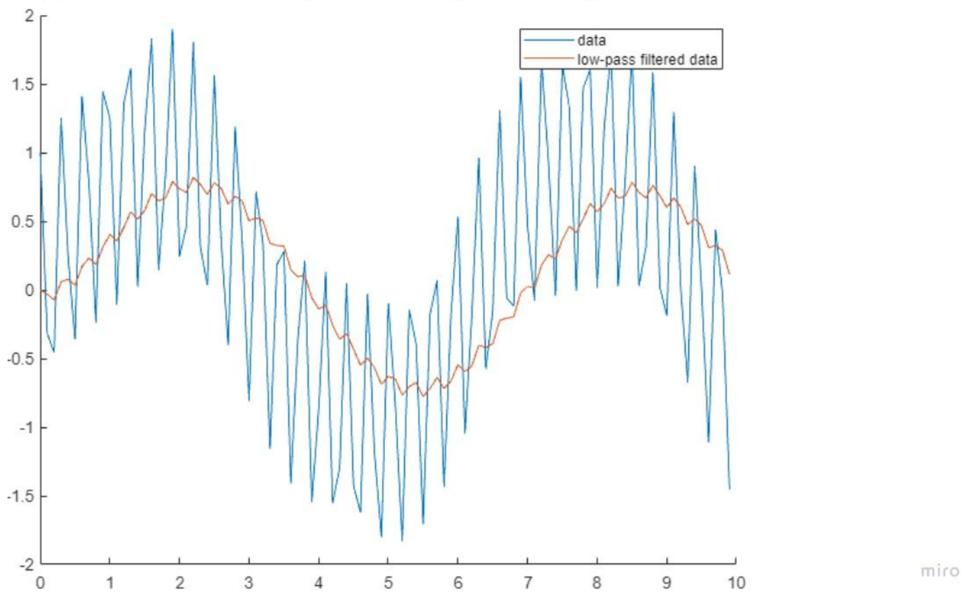
Smoothing a data set is the creation of " an approximating function that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena " ('Smoothing', 2020). Therefore, two types of algorithms can be used for smoothing. There are algorithms using all measurements to obtain the posterior distribution of the acceleration frequently called smoothing algorithms. On another side, there are filtering algorithms that estimate an acceleration at time t using all measurements up to and including time t (Kok et al., 2017). In our case, we need to determine the acceleration

estimation in run-time. Thus, we can't wait for all measurements and we need to use filtering algorithms.
In pose estimation, some popular filtering algorithms are the low-pass filter, the high-pass filter, the Kalman filter and the extended Kalman filter.

1.3.2.1 Low-pass / High-pass filter

"A low-pass filter is a filter that passes signals with a frequency lower than a selected cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency." ('Low-Pass Filter', 2021)
A frequency is the variation of a signal. Low frequency represents a low variation of the signal and high frequency represents a high variation of the signal.
To simplify, the low-pass filter captures the general shape of the data, reducing sudden variations of the data.

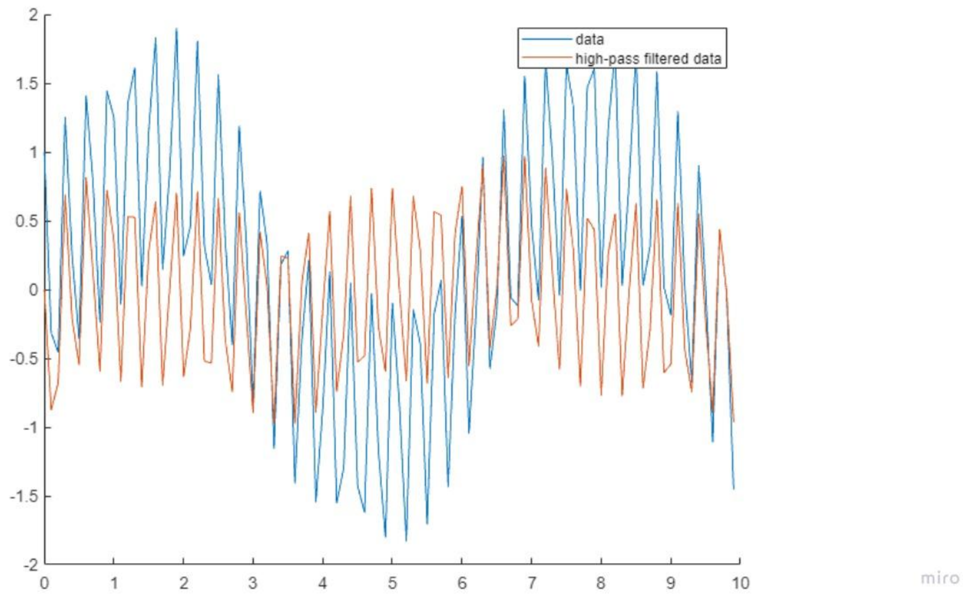
Figure 1.2.3.1.A : Exemple of a low-pass filter applied on data



It is used mainly in audio processing to attenuate sound. It can also be used in image processing to blur a picture. Lastly, it is used in electronics to reduce the noise of the signal.
Different algorithms exist to apply a low-pass filter. The main difference between low-pass algorithms is the difference between the low and high frequencies. For example, the Ideal low-pass filter cuts all frequencies higher than the cutoff frequency, RC low-pass filter reduces frequency linearly after the cutoff and the Butterworth filter reduces frequencies with a smoothstep at the cutoff frequency.

On the opposite, “A high-pass filter (HPF) is an electronic filter that passes signals with a frequency higher than a certain cutoff frequency and attenuates signals with frequencies lower than the cutoff frequency” (‘High-Pass Filter’, 2021)
To simplify, the high-pass filter captures only the important modification of the data and reduces constant modification.

Figure 1.2.3.1.B : Exemple of a high-pass filter applied on data



It is used in audio processing to avoid low frequency which could interfere or damage a speaker. It is also used in image processing to detect the edge of an image. In our case, it can be used to remove drift and only capture the variation of the signal.
The same algorithm as the low-pass filter exists for the high pass filter but they are reversed to capture mainly high frequency.

1.3.2.1 Kalman Filter and Extended Kalman Filter

The Kalman Filter is an algorithm allowing to estimate a value from a measured value and to estimate its probability.
It is mainly used in navigation for dead-reckoning because it allows to make a prediction and update it thanks to new values (Cooper & Durrant-Whyte, 1994).
The algorithm is based on a model and estimates the value from the model.
First, it estimates a value from the measured value, then it obtains the next measured value and compares it to the estimated value. The next step is to calculate the estimation accuracy and use it to make a new estimation. It will restart with a new measured value.

The main difficulty is to define the correct base model and the Kalman filter is linear and can only predict data moving linearly.

Another version of the Kalman Filter is the Extended Kalman Filter (EKF). It modifies the Kalman Filter in order to be non-linear. The purpose is to predict the linearity of a specific region of the data and change this linearity over time.

The main advantage of the Kalman Filter is its multidimensional property. Indeed, Kalman Filter can use vectors of input and calculate vectors of output using matrices. Another important advantage of the Kalman Filter is the possibility to obtain the uncertainty of a value.
The main disadvantage of the filter is the based model reducing the detection of sudden movement.

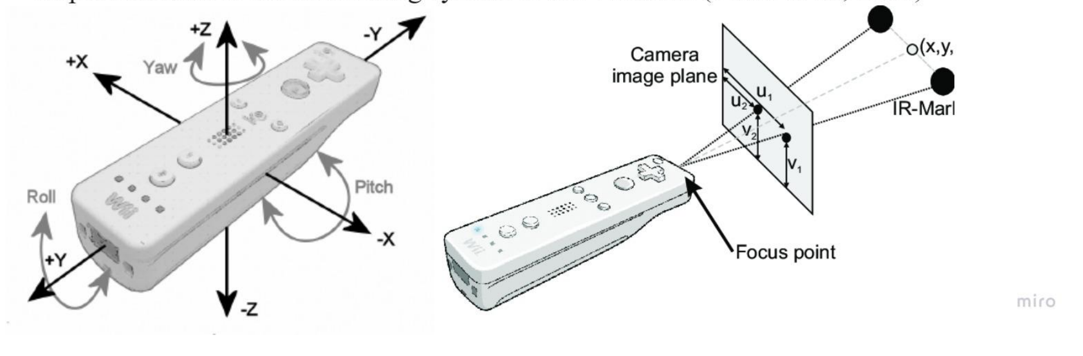
1.3.3 Drift Correction

The filtering algorithms are implemented in order to obtain more accurate values from measurement values. However, even with a filter algorithm, the value will not be totally accurate, and as described in 1.2.2, a small inaccuracy in acceleration can occur in an important drift. Therefore, it is important to implement a way to correct the drift on run-time.

1.3.3.1 Sensor Fusion

The most standard solution is the use of another sensor. In many cases, this sensor is the optical sensor and a popular example of this technology is the WiiMote. The WiiMote uses the inertial sensor to detect acceleration when it is not pointing at the sensor bar. When the WiiMote is pointing at the sensor bar, the camera inside detects the infrared of the sensor bar and can calculate the position and orientation of the WiiMote avoiding drift position (Petric et al., 2010).

Figure 1.3.3.1.A : Disposition of the Wii Remote's accelerometers. (left) (Ubilla & Cadiz, 2010) Representation of the IR tracking system of the WiiMote (Petric et al., 2010)



Another popular sensor fusion is the use of a magnetometer. It is frequently used to determine the orientation of the device based on the orientation of the magnetic field and more precisely the earth magnetic field.
Then, for outdoor pose estimation, the GPS is also a sensor used to confirm the position of the device on the earth (Markovska & Svensson, 2019).
For indoor pose estimation, some experiments use Wifi to determine the position of the device (Lindner et al., 2004).

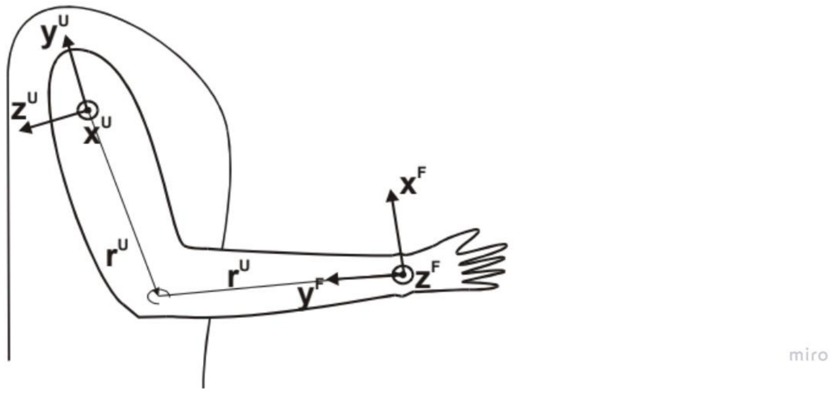
1.3.3.2 Use of the model motion information

In some cases, the model motion of the device is known and this knowledge can help the algorithm on the correction of the drift.

A strategy is the detection of a pattern allowing the algorithm to create a pseudo-measurement of zero velocity (Markovska & Svensson, 2019). It is often used for step detection on foot-mounted IMU. When the IMU tracks the position, it will use a step detection algorithm to recognize the states of the gait cycles in order to apply the zero velocity update when the foot is on the ground.

Another strategy is the detection of restrictions. A common motion model with restriction is the arm motion model (Shen et al., 2018). A human's hand location must always lie on the surface of a sphere centred at the elbow. With this restriction, it is possible to correct the location estimation. If the hand location is too far away from the elbow, it is possible to recalibrate the position estimation.

Figure 1.3.3.2.A : The definitions of the segment reference frame. The unit y axes are defined along the segment, upwards. In anatomical position, the z axes point in the dorsal direction and the x-axes laterally. (Luinge, 2002)



1.4 Use of IMU on a program

To use inertial sensors, it is necessary to access the data. First, data are processed by the device OS. Secondly, data are processed by the library API of the engine. Thus, we will see some of the most popular devices for inertial sensors and then we will see how we can use them on an engine.

1.4.1 Inertial sensors from the device

A popular tool used for the research on inertial sensors is the Arduino. The Arduino is defined as “an open-source electronics platform based on easy-to-use hardware and software” (*Arduino - Introduction*, n.d.). It is an easy way to program the interaction of a variety of inputs such as inertial sensors.
In this paper, we will concentrate on smartphone API. The two most popular smartphone OS are Android and IOS. They both receive the raw data from the sensor and use it through an API. In addition to raw data, they also provide computed data.
In IOS, a class MotionData allows the user to access processed device motion data such as the attitude, an unbiased rotation rate, the gravity and the acceleration without gravity (Allan, 2011, p.72).
In Android, there is also processed sensor data available such as the gravity, the linear acceleration or the rotation matrix of the device (Milette & Stroud, 2012, p.92-98).
From this step of this study, the research will be based on an Android device. It is a limiting factor for the study but it eases the research and allows to go further on the practical dimensions of the study.

1.4.2 Engine interacting with IMU

There are several engines allowing the user to interact with inertial sensors such as the Unreal Engine or Android Studio, but the most popular engine for mobile video games on Google Play is Unity3D (Liu et al., 2018).
Unity3D is a cross-platform game engine developed in 2005. It allows developers to interact with games using C# scripts.
Motion sensors in Unity are divided into 3 displays, the accelerometer returning only the acceleration, the Compass returning the magnetic heading, and the raw geomagnetic data and the gyroscope returning the gravity acceleration vector, the rotation rate, the rotation rate unbiased, the user acceleration and the attitude in Quaternion(*Unity - Scripting API: Input*, 2020). The Quaternion is a vector

composed of 4 variables (x,y,z,w) based on complex numbers. It is often used to describe an attitude because it is compact, and doesn't suffer from gimbal lock (*Unity - Scripting API: Quaternion*, 2020).

1.5 Summary of findings

By doing this research, it is possible to determine the main issues of the project.
The first issue is the multiple integrations for acceleration and angular speed which will occur in a drift.
The second issue is the missing of external anchors, avoiding the drift to be corrected easily and causing an important disparity between the estimated value and the real value.

For this, some solutions have been developed over the years.
The first step is to determine the measurement model. The measurement model is the parameter added to a measure in order to obtain the real value. In our case, the measured acceleration at which the bias and noise are removed.
The bias can be removed with calibration, calculating the standard bias from a known orientation and position and removing it from the measure.
The noise can be removed using a smoothing algorithm, an algorithm estimating the value based on the previously estimated values and the measured value. There is a diversity of filters depending on the use of the smoothing.
However, an estimated value reduces but does not avoid drift. Therefore, some processes have been used over the years. A first process is the use of an external sensor such as Wifi or GPS, A second process is the detection of a known bunch of data such as sep detection. A third process is the detection of an impossible value when the motion model of the data is known.

These researches highlighted the fact that the development of an algorithm of dead reckoning depends a lot on the use of this algorithm because it will determine which smoothing algorithm will be used and which drifting correction mechanism will be developed.

Another important point is the multiple tools available from the smartphone API and which one should be used.

2 Qualitative part

In order to stay close to modern projects, it is essential to analyze some position estimation projects and which solutions they are using.

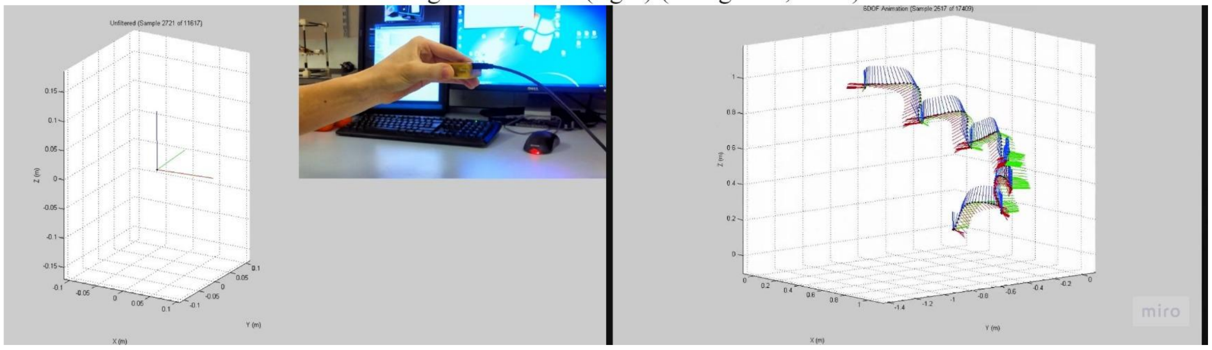
2.1 How project were selected

In order to compare them in the best way, it is necessary to use similar projects. Thus, the project has to use an accelerometer and a gyroscope to obtain a position estimation and an orientation estimation and has to be body-based. In order to perform better analyses, the project has to have a description of the algorithms used and why it uses them. And finally, it has to prove the efficiency of their algorithms with some tests.

2.1.1 X-IMU Oscillation / Gait tracking

The first project is made up of two programs developed in 2014 by the company x-io Technologies for their device x-IMU (Madgwick, 2013).
The first program is the tracking of a foot during a movement over floors and stairs and the second program is the tracking of the position and orientation of the device during an oscillatory movement.
The project provides their source code for MatLab and two videos of the demonstration of their algorithms.

Figure 2.1.1.A : Screenshots from the Oscillatory motion tracking with x-IMU (left) and the Gait tracking with x-IMU (right) (Madgwick, 2013)


Unfiltered (Sample 2721 of 11617)
Z (m)
X (m)
Y (m)
6DOF Animation (Sample 2517 of 17409)
miro

2.1.2 Using Inertial Sensors for Position and Orientation Estimation

The second project is a university paper published in 2017 on Foundation and Trends (Kok et al., 2017). Foundations and Trends in Signal Processing is a journal covering the technologies/fields/categories related to Signal Processing (Q1) (*Foundations and Trends in Signal Processing | Resurchify*, 2021).

These projects describe all the steps and decisions made during the development of an algorithm to track the position and orientation using the IMU of a smartphone.
It also provides an analysis of their tests.

2.1.3 Position Estimation with a low-cost Inertial Measurement Unit

The third project is a paper published in 2018 by the University Pompeu Fabra (Llorach et al., 2014).
This project uses the IMU of an Oculus Rift virtual reality headset in order to track the head position.
It presents the steps of the position estimation and drift correction of their algorithm.
It also provides an analysis of their tests.

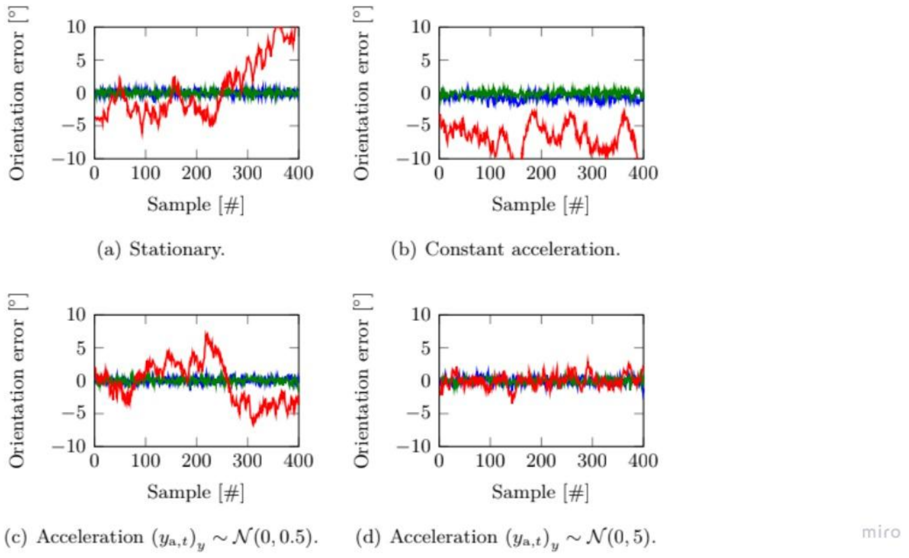
2.2 Project Analysis

All these projects use different algorithms because the purpose of their use is different.
The Gait program of the x-IMU doesn't use any filter for position estimation because all the drift is avoided by the step detection.
The Oscillatory program of the x-IMU uses two high-pass filters on the velocity and position to remove the drift caused by the integrations.
The second project uses an Extended Kalman Filter for orientation estimation and position estimation because it uses all the values to directly obtain a point estimation and an associated measure of uncertainty.
The third project uses a low-pass and a high-pass filter on the acceleration to remove noise and a high-pass filter on the velocity to remove the drift.

Furthermore, all these projects use different drift corrections. The second project doesn’t use any drift correction because of the accuracy of the estimation by the smoothing algorithm. The Oscillatory program uses no drift correction either because the high-pass will automatically remove the drift.
The third project uses a basic threshold detection to avoid drift due to noise around zero. The Gait program of the x-IMU uses a step detection algorithm to detect gait and set the velocity at zero when the foot is down.

Finally, it’s necessary to analyze the result according to the tested movement.
For the Gait program, the project displays an effective reproduction of the gait through floor and stairs and, for the Oscillatory program, the project displays an effective reproduction of the oscillatory movement. After testing with custom data, the author detected less effective results because of the use of low-cost sensors of the smartphone compared to the high-performance sensor of the x-IMU. Furthermore, the data are even less accurate for non-gait and non-oscillatory movement.
For the second project, the result depicts less drift for inconstant acceleration than for constant or stationary acceleration as shown in Figure 2.2A
For the third project, according to their estimation, a small amount of displacement can be noticed when the user finishes the movement.

Figure 2.2.A : Orientation errors of the roll (blue), pitch (green) and heading (red) using simulated inertial and magnetometer measurements.(Kok et al., 2017)



2.3 Results of these projects

The first important point from these projects is the influence of the model through estimation. Indeed, the first project depicts a good estimation for the state of their project but a worse estimation when the motion is different. Then, for the second project, its results change according to the type of motion applied to the device.
Another point that came out is the use of different smoothing algorithms. For example, the choice and the efficiency of the smoothing algorithm depend on two points. The first point is the performance of the sensors. If the sensor is

high-performance, it is not necessary to apply a complex smoothing algorithm like in the first project.
The choice of the sensor can also be done depending on the use of the value, such as the second project needs to use a gyroscope, an accelerometer and a magnetometer all together.
Finally, the drift correction is also different depending on the project. The second project using an efficient smoothing algorithm doesn't need drift correction. The drift correction depends also on the type of movement, the Gait program needs an important drift correction based on the step detection, whereas the Oscillatory program doesn’t use any drift correction because it estimates a constantly moving object.

3 Setting up the environment

This part focuses on the definition of the different tests. In order to answer correctly : “How can we improve the position estimation using inertial sensors of a smartphone during fast movements?”, the main objective of this project is to determine whether it is possible, or not, to estimate an accurate motion based on the inertial sensor of a smartphone. It is important to remember that this technique already exists. Before going into several details, it is essential to recall the objectives and the parameters of this work.

3.1 Generic Aspect

The purpose of the project is to develop an algorithm to calculate the position of a smartphone. This position estimation can then be used for motion-based games.
These three points impose some constraints.

3.1.1 Detection of position

The precision of the position will be the comparison between the real value and the estimated value. Because the real value is observed by a unique camera, the position is compared on only 2-axis.

3.1.2 Use of smartphone

The use of a smartphone imposes the use of low-cost inertial sensors to obtain the acceleration and the orientation. Adapting the algorithm to use low-cost sensors is essential to be functional.

3.1.3 Uses in games

The use of this algorithm for motion-based games restricts movements performed by the device. Indeed, it is known that the device will be held by the hand and moved around the body. Furthermore, based on the most popular motion-based game (Jurkovich, 2021), the objective is to detect sports movements such as tennis or saber. It also implies the calculation in real-time with a small delay.

3.2 Focus on precision

The efficiency of the algorithm created is based on the precision of the estimation. The precision is calculated by the difference between the estimation and the real value. The compared value is the position that can be directly observed and must be estimated through the algorithm. Therefore, the level of precision describes the deviation of the estimated position compared to the real position. However, the calculation of the precision must be analyzed through different parameters.

3.1.1 When to calculate the precision

To analyze the precision with the most accuracy possible, it is important to analyze the precision during all steps of the movement. Therefore, the first step is to analyze the precision during the movement to determine its reliability when measured. The second step is to analyze the precision at the end of the movement to obtain the deviation caused by the movement.

3.1.2 What types of movement to observe

As viewed in chapter 2, the types of movement influence the precision of the estimation. In order to optimise the algorithm, it has been decided to test an assortment of specific movements. Because the algorithm is designed to detect movements for a game, the movements have been selected to match with movements of motion-based games. Therefore, the first criterion is the arm based movement, performed using the controller in the hand and the body to be static. The second criterion is to match motion with sports movements such as tennis or saber, which can be described by a quick and large movement with a static start and a static end.

3.3 Description of the environment

Three outside factors influenced the result : Unity, the smartphone and the camera.

3.3.1 Unity

Unity is the game engine used to perform the tests. The Unity version used is the 2020.3.8f1 LTS. The choice of this version has been done because it is a stable version which means the version will remain stable for a long period

(*Unity QA - LTS Releases*, 2021). Furthermore, the version has been released in May 2021, so it's a recent release and represents an accurate state of the engine. Each scenario is tested on an application directly on the smartphone allowing it to record and perform computation directly on the smartphone.

3.3.2 Smartphone

The smartphone used is a Samsung Galaxy S7 bought in 2017. The critical point of this study is the inertial sensors. The accelerometer and the gyroscope are part of the IME K6DS3TR developed by STM. The accelerometer has a resolution of 0.00239 m/s^2 and a maximum range of 78.453 m/s^2. The gyroscope has a resolution of 0.00061 rad/s and a maximum range of 17.4532 rad/s.

3.3.3 Camera

The camera is used to obtain the real position of the smartphone. The camera is a Samsung Galaxy A7 camera recording at 30fps. The video is then obtained in mp4 and processed with Tracker. Tracker is a free video analysis designed for physics education.

3.4 Test protocol

The test protocol is divided into 3 steps : the estimation recording, the real position recording and the comparison.

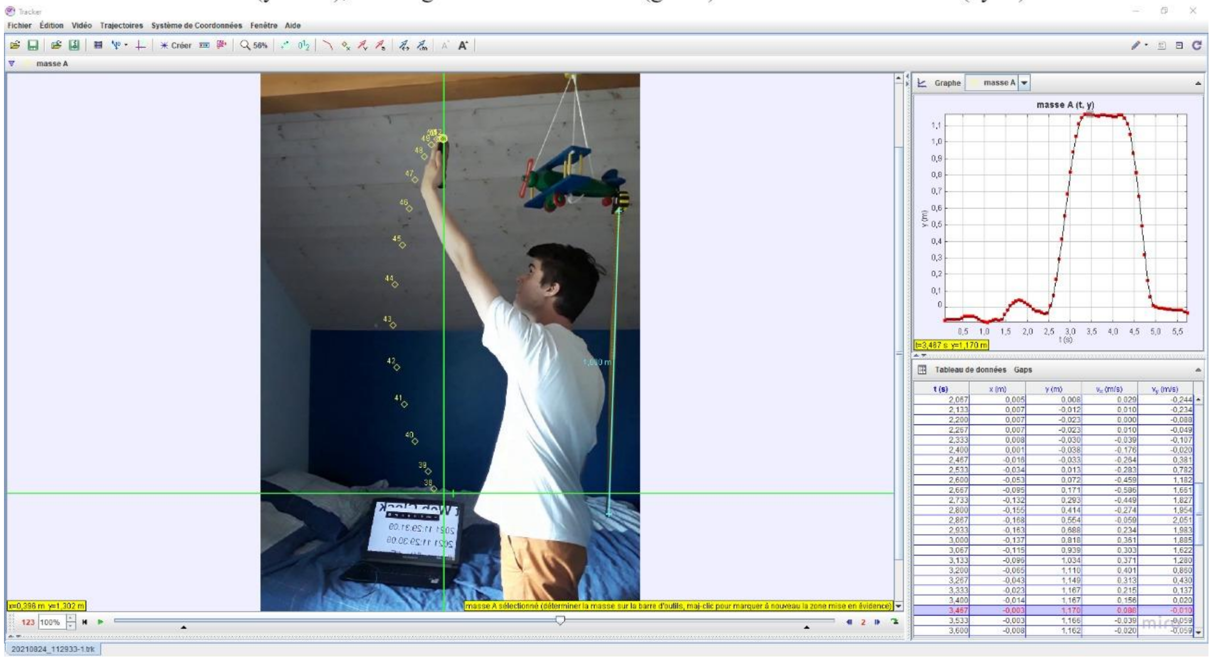
3.4.1 Estimation recording

The estimation is recorded through the Unity application on the smartphone. The application records the real-time estimation using the real-time data from the accelerometer and the gyroscope. At the end of the test, the estimated data is output to a JSON file on the phone.

3.4.2 Real Position estimation

In parallel with the estimation recording, the camera records the movement of the smartphone. The position is then picked by hand through the Tracker to obtain an array of positions and the time-related.

Figure 3.4.2.A : Screenshot of Tracker during the optical tracking with the position of the device(yellow), the origin of the movement (green) and the calibration bar (cyan)


Fichier Édition Vidéo Trajectoires Système de Coordonnées Fenêtre Aide
masse A
Graphe
masse A (t, y)
y (m)
t (s)
t=3,487 s y=1,170 m
x=0,396 m y=1,302 m
masse A sélectionné (déterminer la masse sur la barre d'outils, maj-clic pour marquer à nouveau la zone mise en évidence)
Tableau de données Gaps

t (s)	x (m)	y (m)	v_x (m/s)	v_y (m/s)
2,067	0,005	0,008	0,029	-0,244
2,133	0,007	-0,012	0,010	-0,234
2,200	0,007	-0,023	0,000	-0,088
2,267	0,007	-0,023	0,010	-0,049
2,333	0,008	-0,030	-0,039	-0,107
2,400	0,001	-0,038	-0,176	-0,020
2,467	-0,016	-0,033	-0,264	0,381
2,533	-0,034	0,013	-0,283	0,782
2,600	-0,053	0,072	-0,459	1,182
2,667	-0,095	0,171	-0,586	1,661
2,733	-0,132	0,293	-0,449	1,827
2,800	-0,155	0,414	-0,274	1,954
2,867	-0,168	0,554	-0,059	2,051
2,933	-0,163	0,688	0,234	1,983
3,000	-0,137	0,818	0,361	1,885
3,067	-0,115	0,939	0,303	1,622
3,133	-0,096	1,034	0,371	1,280
3,200	-0,065	1,110	0,401	0,860
3,267	-0,043	1,149	0,313	0,430
3,333	-0,023	1,167	0,215	0,137
3,400	-0,014	1,167	0,156	0,020
3,467	-0,003	1,170	0,088	-0,010
3,533	-0,003	1,166	-0,039	-0,059
3,600	-0,008	1,162	-0,020	-0,059

3.4.3 Position Comparison

The two data are then displayed by a Javascript program allowing to visualize the graph of the two data. The result will contribute to highlight the precision of the algorithm.

3.5 Summary of tests

The main purpose of the tests is to verify the possibility of using the inertial sensor of a smartphone to estimate its motion. Once the algorithm is developed, it is necessary to verify its efficiency with movement similar to its use.

The efficiency is verified by comparing the precision of the estimated position and the real position.

However, the precision depends significantly on the movement performed. Based on the motion-based game, the movements are quick with a static start and a static end.

The last point is the comparison between the estimation and the real movement. It is done using a camera to record all the positions of the device during the movement.

4 Implementation and results

The implementation of the algorithm is based on the research and all the similar project analysis. With this information, the first step was to analyze the raw data, then to develop diverse algorithms to analyze which one is the most efficient for the aimed objective and finally test these algorithms to question their efficiency.

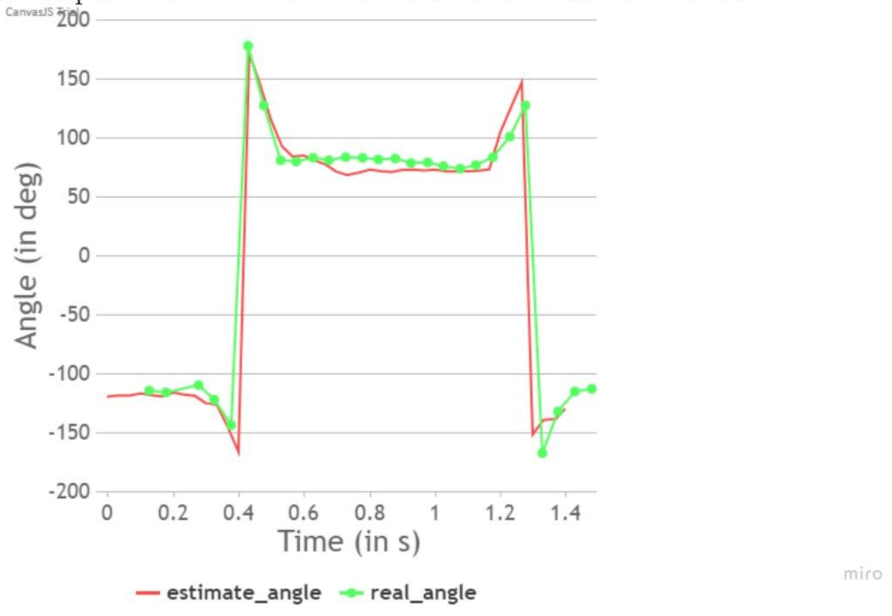
4.1 Analysis of the data

The implementation of the algorithm is developed on Unity. Therefore, the first step is to analyze the sensor data provided by the API.

4.1.1 Orientation

The orientation is provided by 3 values. The rotation rate and the rotation rate unbiased provide the angular speed detected by the gyroscope. The attitude is the estimation of the rotation provided in Quaternion. According to the author, the factors used for this purpose are the rotation rate and the gravity. After some tests, it has been determined that the attitude is accurate enough to be used for the estimated orientation of the device. The error during the movement is indeed lower than 15° and the error at the end is also lower than 15° and can be automatically adjusted after some time.

Figure 4.1.1.A : Representation of the real and estimated rotation of the device



4.1.2 Acceleration

The acceleration is also provided by 3 values. The acceleration is the raw value obtained by the accelerometer. From this value and the gyroscope, Unity provides an estimation of the gravity. Finally, subtracting the gravity from the acceleration, Unity provides the user acceleration. It is this value that will be used to estimate the position. However, this value can't provide an accurate approximation. As seen during the state of the art (chapter 1), the value of the accelerometer can't be used directly. In fact, as seen in Figure 4.3.3.C, an inaccuracy can be observed in these values. With raw value, it can observe after 5 seconds an inaccuracy of the speed greater than 1m/s that occurs a constant drift on the position estimation. Therefore, it is necessary to apply some algorithms before using these data.

4.2 Development of the algorithms

4.2.1 From Local to Global

The first step before using the acceleration is to modify it to obtain the acceleration on the global space. As explained in 1.2.2, the acceleration obtained by the accelerometer is, indeed, related to the device axis. To obtain it in accordance with the room axis, it is necessary to rotate it with the device orientation.
The device orientation is obtained by the attitude. The attitude is a Quaternion. But this quaternion is not oriented in the Unity space. Therefore, the quaternion is modified to convert right-handed (X axis to the right) to left-handed (X axis to the left).

Figure 4.2.1.A : Equation of transformation from right handed Quaterion to left handed quaternion

$$Q_{left} = f\left(Qright \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \right) = \begin{bmatrix} -x \\ -z \\ -y \\ w \end{bmatrix}$$

where Qright is the obtained attitude and Qleft is the converted attitude and f is the conversion function.

Furthermore, the values provided by the accelerometer are also not in the same space. The value is modified to convert from MEMS space to Unity space.

Figure 4.2.1.B : Equation of transformation from MEMS acceleration to Unity acceleration

$$A_{unity} = f\left(A_{mems}\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} -x \\ -z \\ -y \end{bmatrix}$$

miro

where Amems is the acceleration from the accelerometer and Aunity is the converted attitude and f is the conversion function.

The user acceleration is then multiplied by the attitude. For this, the attitude is transformed into a rotation matrix. After the rotation, the acceleration is obtained in global space. The last step is to convert from g to m/s^2. Because the MEMS provides the acceleration in g and 1g is equal to $9.81m/s^2$ on earth, the acceleration is multiplied by 9.81 to be obtained in m/s^2.

4.2.2 Calibration

Based on the first chapter, the first step was to check if calibration is necessary. After some testing and analysis on the acceleration, it can be observed that the acceleration doesn't have any constant bias.

4.2.3 Drift Correction

The second step was a drift correction. The first solution is the implementation of a pseudo-measurement of zero velocity. Indeed, because the detected movement is quick and short, it can be assumed that if the acceleration is small, the velocity can be estimated to be null. Therefore, the acceleration is analyzed during a certain duration, and if the sum is null for all the duration, the acceleration is then directly reset to zero.

Figure 4.2.3.A : Algorithm of zero-based drift correction

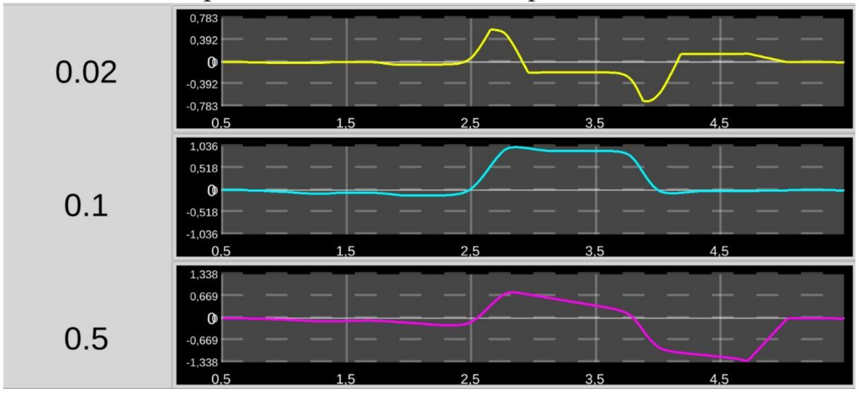
$$if\left(\sum_{k=t-\Delta}^{t}(Acc_k) = 0\right) then\ Acc_t$$

miro

where Acc_t is the acceleration at time k on each axis, t is the current time and Δ is the analysis window.

After the implementation, it has been determined that a duration of 0.1 seconds allows the observation of an efficient drift correction according to certain movements. Indeed, if the duration is too small, the drift correction will be applied too early and the movement will be modified and if the duration is too large, the drift correction will be applied too late.

Figure 4.2.3.B : Comparison between three delta parameters for the reset velocity



Another drift correction has been implemented using RC Highpass Filter. With the high pass filter implemented on the velocity or the position, it can eliminate the constant drift of the integration.

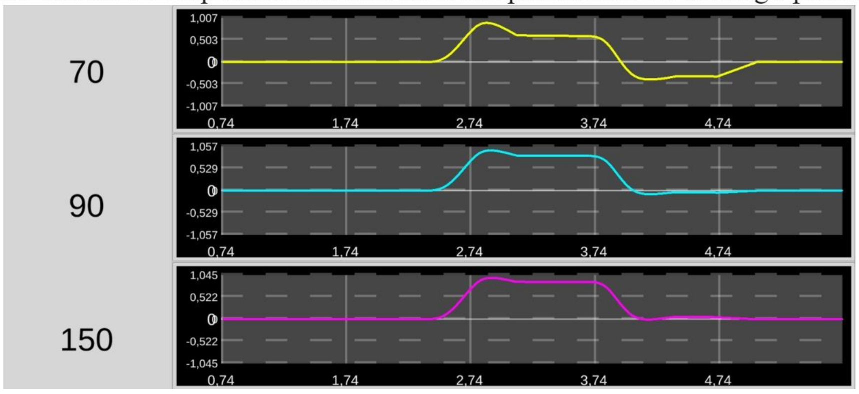
Figure 4.2.3.C : Equation of RC High-pass filter

$$\alpha = RC \,/\, (RC + \Delta)$$
$$eacc_t = \alpha * eacc_{t-1} + \alpha * (acc_t - acc_{t-1})$$

where RC is the High-pass parameter, Δ is the delta time between two frames, t is the current time, $eacc_t$ is the estimated acceleration at time t, $eacc_{t-1}$ is the previous estimated acceleration, acc_t is the measured acceleration at time t, acc_{t-1} is the previous measured acceleration.

After some tests, it has been determined that an RC to 90 allows an efficient drift correction without excessive loss of value, according to the accuracy of the data received.

Figure 4.2.3.D : Comparison between three RC parameters for the High-pass filter



.

4.2.4 Smooth algorithm

The last step of the algorithm is the smooth algorithm to remove the noise of the data. A simple algorithm is the implementation of a threshold to remove noise around zero. It has been implemented to remove noise on the acceleration.

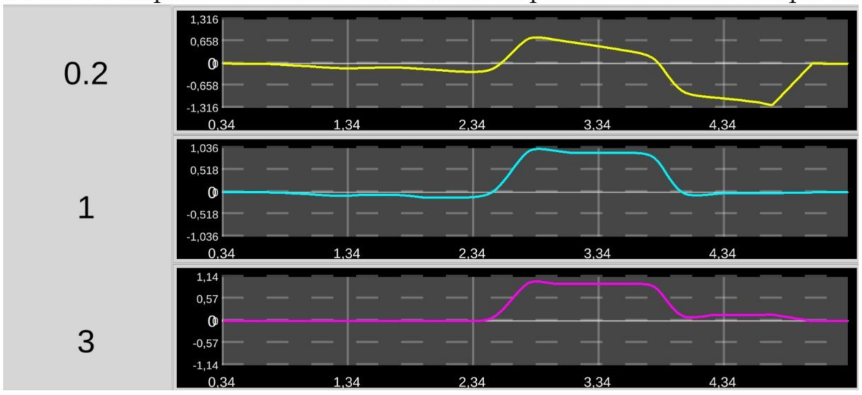
Figure 4.2.4.A : Algorithm for simple smoothing

$$if\ (|Acc|\ >\ Th)\ then\ Acc\ =\ 0$$

where Acc is the acceleration on each axis and Th is the threshold

After some tests, it has been determined that a threshold to 1 m/s² allows efficient smoothing without excessive loss of value, according to the accuracy of the data received.

Figure 4.2.4.B : Comparison between three threshold parameters for the simple smoothing



After this, an implementation of the Kalman filter has been applied on each axis of the acceleration. The Kalman filter works with 2 values, the noise (Q) and the uncertainty of the measurement (R).

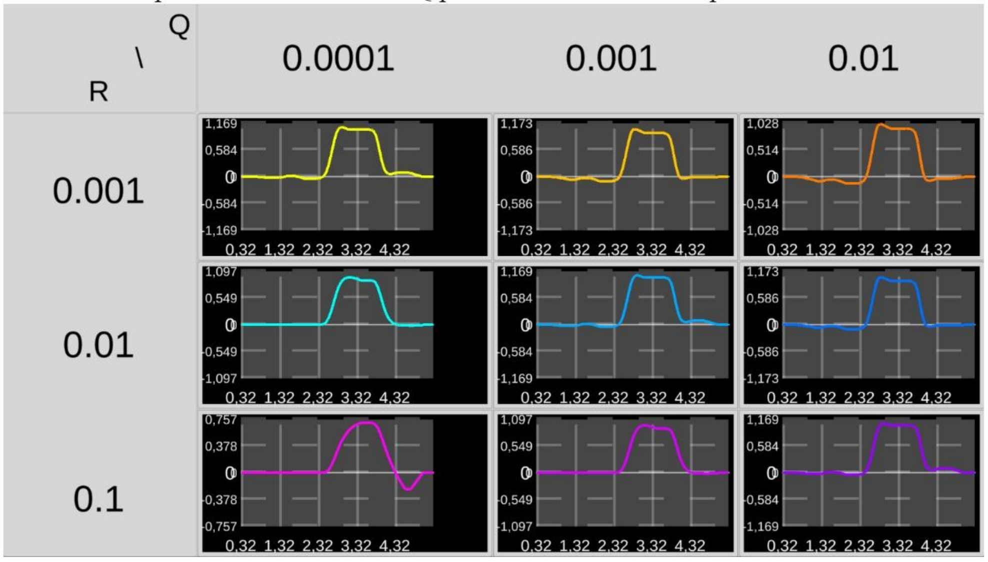
Figure 4.2.4.C : Algorithm of the Kalman filter

$$K\ =\ (p_{n-1}\ +\ Q)\ /\ (p_{n-1}\ +\ Q\ +R)$$
$$p_n\ =\ R * (p_{n-1}\ +\ Q)\ /\ (R + \ p_{n-1}\ +\ Q)$$
$$x_n\ =\ x_{n-1}\ +\ (m\ -\ x_{t-1}) * K$$

where K is the Kalman Gain, p_{n-1} is the previous uncertainty in estimation, R is the uncertainty of the measurement, Q is the noise, x_{n-1} is the previous estimation and m is the measurement

After multiple tests, it has been determined that the best parameters are Q=0.001 and R = 0.01.

Figure 4.2.4.D : Comparison between three Q parameters and three R parameters for the Kalman filter



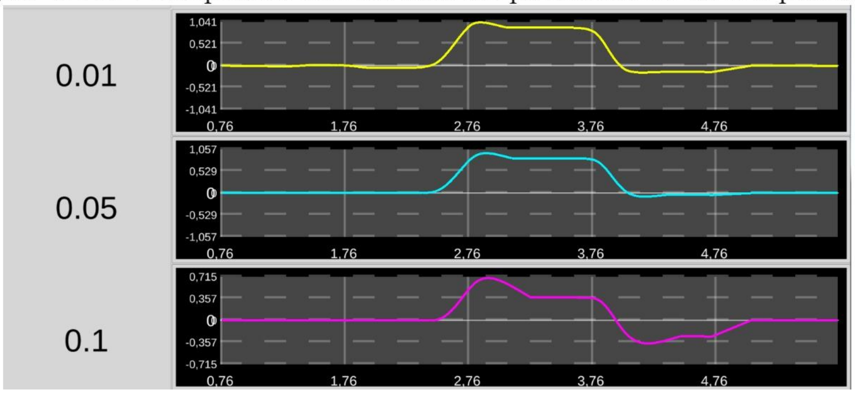
Another smooth algorithm implemented is the low-pass filter. It is applied to the acceleration to remove the noise.

Figure 4.2.4.E : Equation of RC Low-pass filter

$$\alpha = \Delta \ / \ (RC + \Delta)$$
$$eacc_t = \alpha * acc_t + (1 - \alpha) * eacc_{t-\Delta}$$

After some tests, it has been determined that an RC to 0.05 occurs an efficient smoothing.

Figure 4.2.4.F : Comparison between three RC parameters for the low-pass filter



4.2.5 Review of algorithm

Three algorithms will be analyzed during the tests.

The first algorithm is a simple algorithm with the simple smooth algorithm and a drift correction.

Algorithm 1 Position estimation with simple smoothing and drift correction

Inputs : the global acceleration a, the delta time dt, actual timer t, a threshold parameter for smoothing th, a time delta for pseudo-zero velocity d
Outputs : the estimation of the velocity ev, the estimation of the position ep

```
1. For each axis of a
        a. Simple smooth, see Figure 4.2.4.A
2. Integration
        ev = ev + a * dt
3. For each axis of a
        a. Drift Correction, see Figure 4.2.3.A
4. Integration
        ep = ep + v * dt
```

The parameter of the drift correction applied to this algorithm is a threshold to 1 m/s^2 and a delta to 0.1.

The second algorithm is a Kalman algorithm with a Kalman filter for smoothing and a drift correction.

Algorithm 2 Position estimation with Kalman filter and drift correction

Inputs : the global acceleration a, the delta time dt, actual timer t, a noise parameter Q, uncertainty of the measurement R, a time delta for pseudo-zero velocity d
Outputs : the estimation of the velocity ev, the estimation of the position ep

```
1. For each axis of a
        a. Kalman filter, see Figure 4.2.4.C
2. Integration
        ev = ev + a * dt
3. For each axis of a
        a. Drift Correction, see Figure 4.2.3.A
4. Integration
        ep = ep + v * dt
```

The parameter of the drift correction applied to this algorithm is a threshold to 1 m/s^2 and a delta to 0.1.

The parameters of the kalman filter are $Q = 0.001$ and $R = 0.01$.

The third algorithm is an RC low-pass filter for smoothing, a drift correction and an RC high-pass filter.

Algorithm 3 Position estimation with RC High-pass and Low-pass filter and drift correction

Inputs : the global acceleration a, the delta time dt, actual timer t, a parameter for low-pass filter RClow, a parameter for high-pass filter RChigh, a time delta for pseudo-zero velocity d
Outputs : the estimation of the velocity ev, the estimation of the position ep

```
1. For each axis of a
        a. Low-Pass filter, see Figure 4.2.4.E
2. Integration
        $ev = ev + a * dt$
3. For each axis of a
        a. Drift Correction, see Figure 4.2.3.A
4. For each axis of ev
        a. High-Pass filter, see Figure 4.2.3.C
6. Integration
        $ep = ep + v * dt$
```

The parameter of the drift correction applied to this algorithm is a threshold to 1 m/s^2 and a delta to 0.1.
The parameter of the RC lowpass filter is $RC = 0.5$ and for the RC highpass filter $RC = 90$.

4.3 Results

Before analyzing the different results, it is important to recall the evaluation criteria and the test environment.

4.3.1 Review of evaluation criteria

Two points must be analyzed on the results. First of all, the precision during the movement. The movement must be recognizable and the most accurate possible. Secondly, the precision at the end of the movement must be analyzed. The end of the movement must be static with no end drift and with the best precision. Even if the test is focused on the main movement, it is also interesting to check the estimation on the other axis during the movement.

4.3.2 Description of the tests

The tests are based on 3 movements. The first movement is a lateral translation with no rotation during the movement, the second movement is a horizontal translation with no rotation during the movement and the last movement is a horizontal arc with a rotation during the movement of approximately 180°.

There are two tests by movement, one “fast” and one about 2 times slower. These two tests aim to show the difference of the position estimation according to the speed.

Each test has been performed with the same sequence. During each test, there is a back and a forth. The back and the forth are separated by a static pause of approximately 0.5-1 second. The start and the end of the movement are also static for approximately 0.5-1 second.

During these tests, 5 data will be analyzed. The first data is the real data obtained by the camera. It provides the real position of the device on axes X and Y. This data is used to analyze the accuracy of the estimated position. The real data provide the position approximately every 0.07s with a precision of 0.005m on static and 0.05m on movement.

The second data is the raw data of the accelerometer in m/s^2 with the rotation around the attitude applied. These data allow the observation of the inaccuracy of the based data which cause a drift.

The third data is the result of the raw data processed with drift correction and a simple smooth algorithm described in Algorithm 1. The fourth data is the result of the raw data process with drift correction and the Kalman FIlter described in Algorithm 2. The fifth data is the result of Algorithm 3 which processes the raw data through drift correction and high-pass and low-pass filters. All the data provided by the application are processed approximately every 0.016s.

4.3.3 Lateral translation

The first movement is a lateral translation. According to the real data, the first test is a translation of 0.5m on the axis X at approximately 2.75 m/s. The second test is also a translation of 0.5m on the axis X at approximately 1.3 m/s. In the two cases, there is a first translation of 0.3m on the axis X at approximately 0.75 m/s negligible because done for the placement. For the two tests, the translation in Y is negligible because it is less than 0.25m.

Figure 4.3.3.A: Representation of the observed position of the device during the lateral movement throughout the first test (left) and the second test (right)

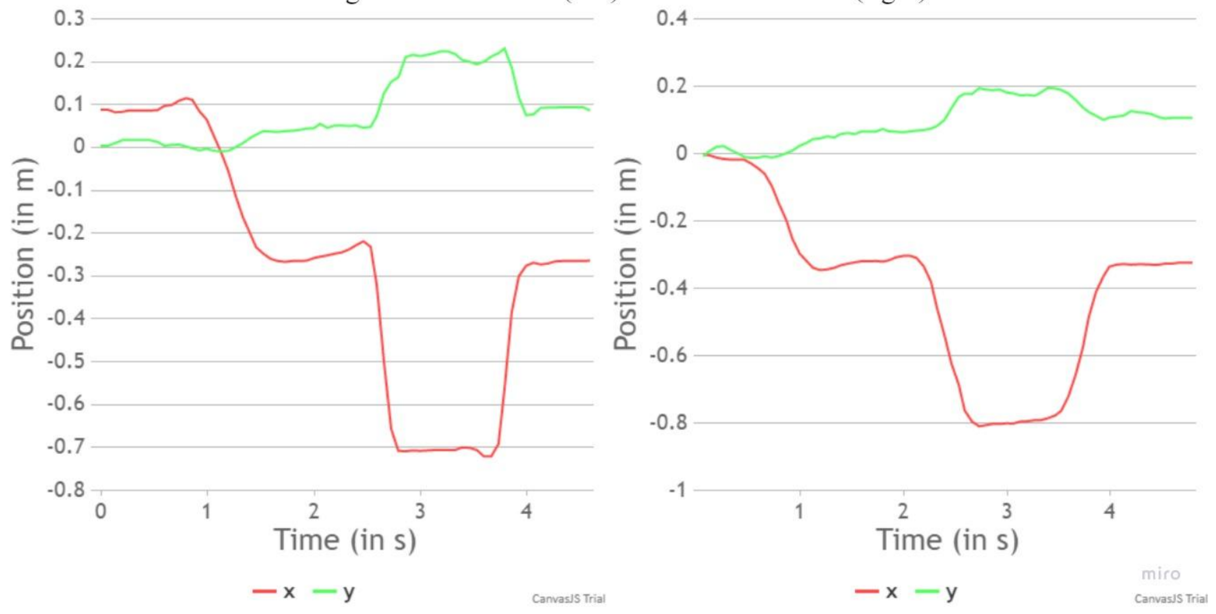
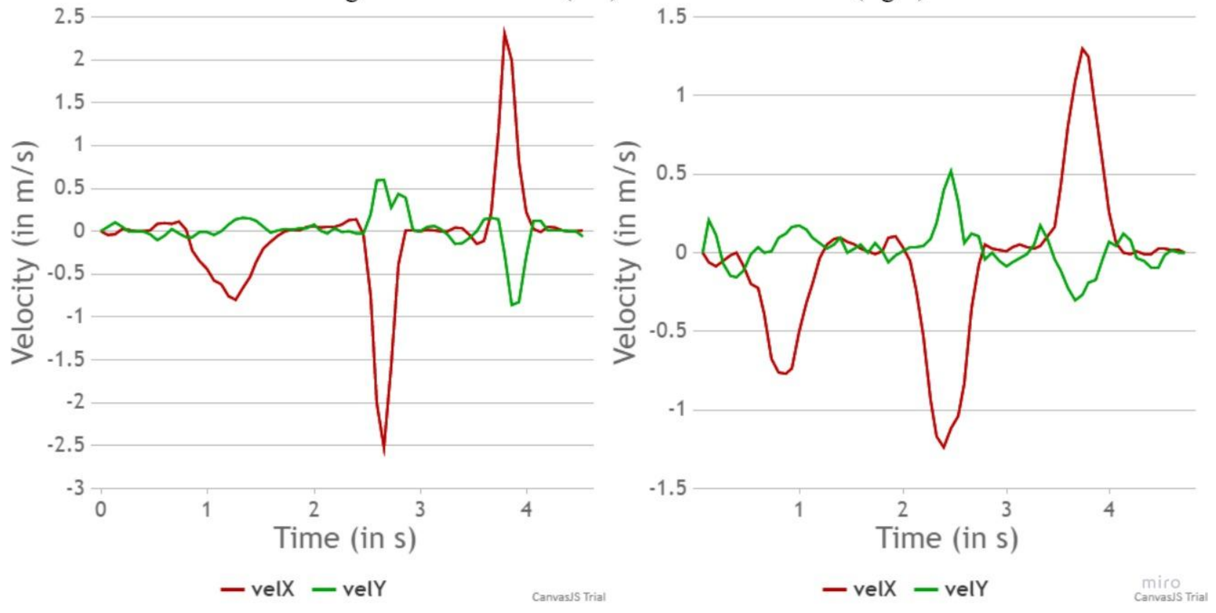


Figure 4.3.3.B : Representation of the observed velocity of the device during the lateral movement throughout the first test (left) and the second test (right)



When analyzing the raw data, we can observe a drift of 2 m/s for the first test and 1.5 m/s for the second test. Indeed, due to the noise of the acceleration, the velocity is not reset at zero and is stabilized at 2 m/s and 1.5 m/s. But it is still possible to recognize the movement during the drift. The end of the drift is estimated at 5m on the first test and 10m on the second test. We can also observe a drift on the axis Y of 0.5 m/s for the first test and 1.75 m/s for the second test.

Figure 4.3.3.C : Representation of the estimated position during the lateral movement throughout the first test (left) and the second test (right)

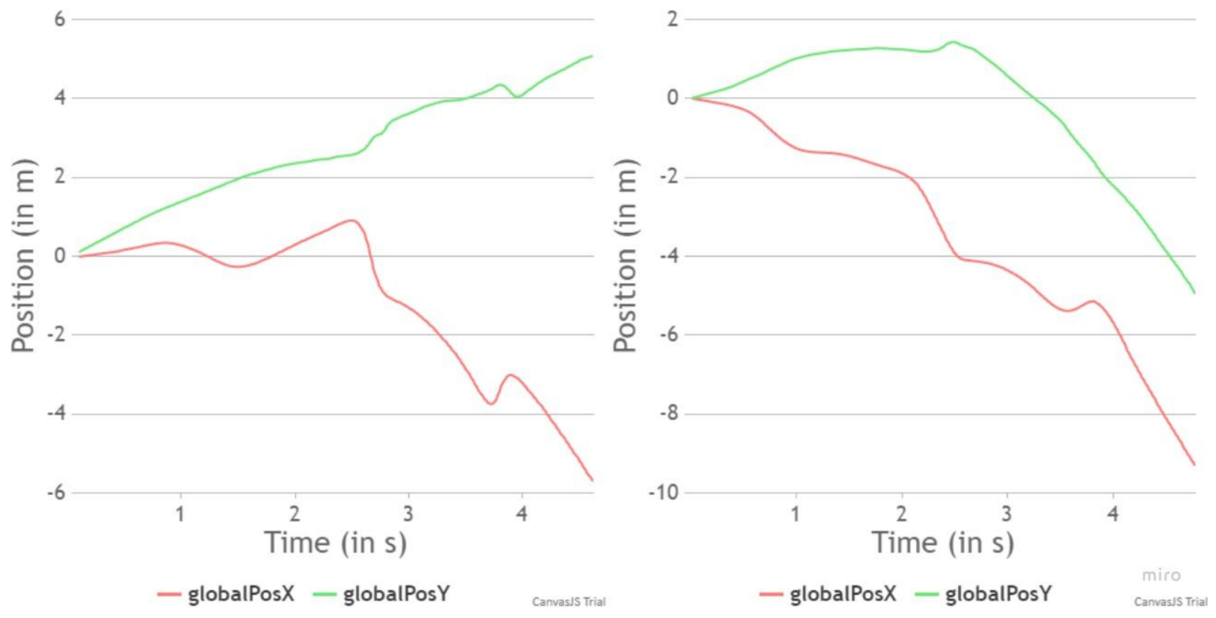
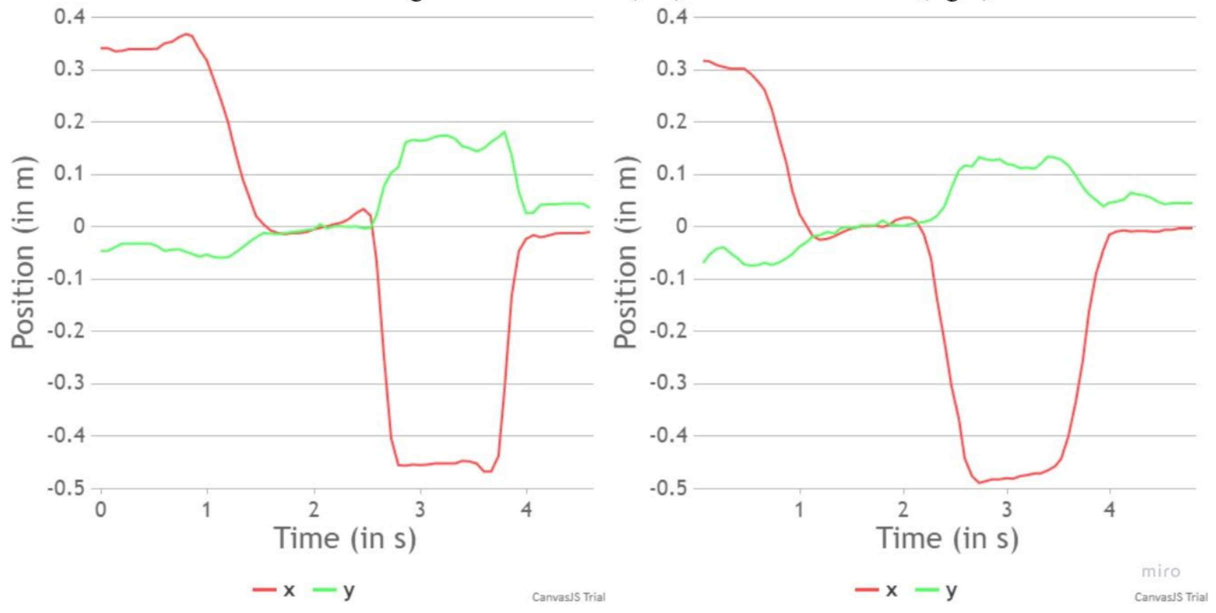
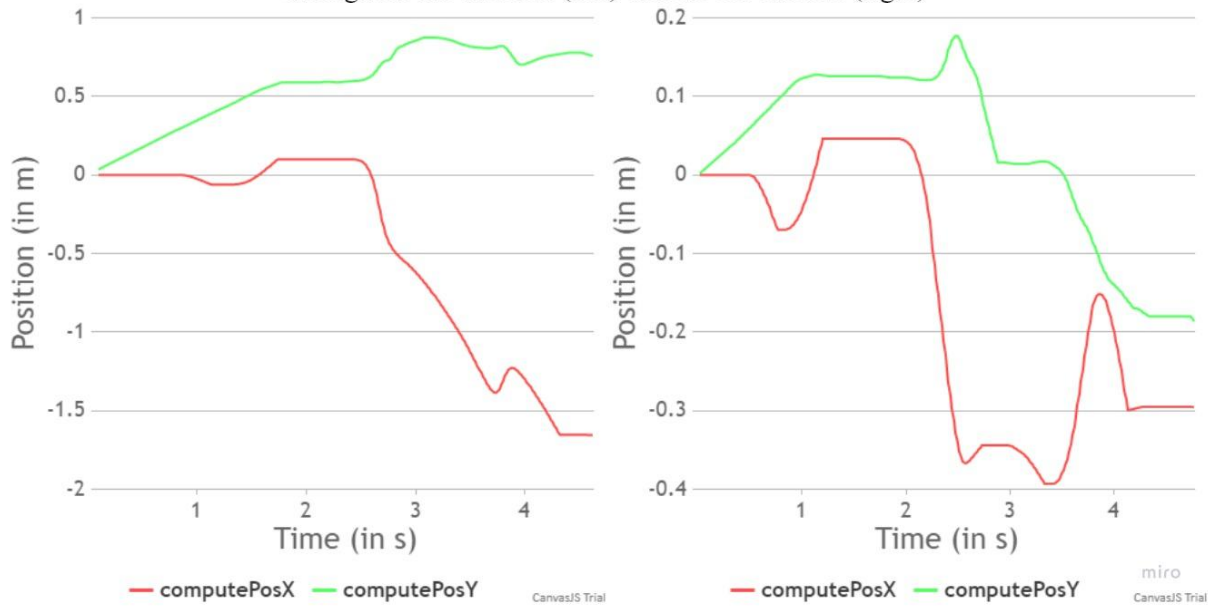


Figure 4.3.3.D : Representation of the re-centered observed position of the device during the lateral movement throughout the first test (left) and the second test (right)



The data from the first algorithm provide some improved estimations. But on both tests, the first translation isn't accurate due to the movement being too small, so the author will use the data with an origin at the start of the main movement as shown in Figure 4.3.3.D. For both tests, the detection during the first translation is accurate with the real data. But for the first test, there is no drift correction due to too much noise during the pause. Therefore, the drift is increasing with an average of 1 m/s. The delta at the end is 1.3m.
For the second test, there is a correct drift correction with a delta during the pause of 0.1m. But, during the second movement, the movement is inaccurate. Indeed, the speed decreases too much due to the inaccuracy of the raw data. The delta at the end of 0.3m. Finally, on the Y axis, there is an important drift at the start and at the end.

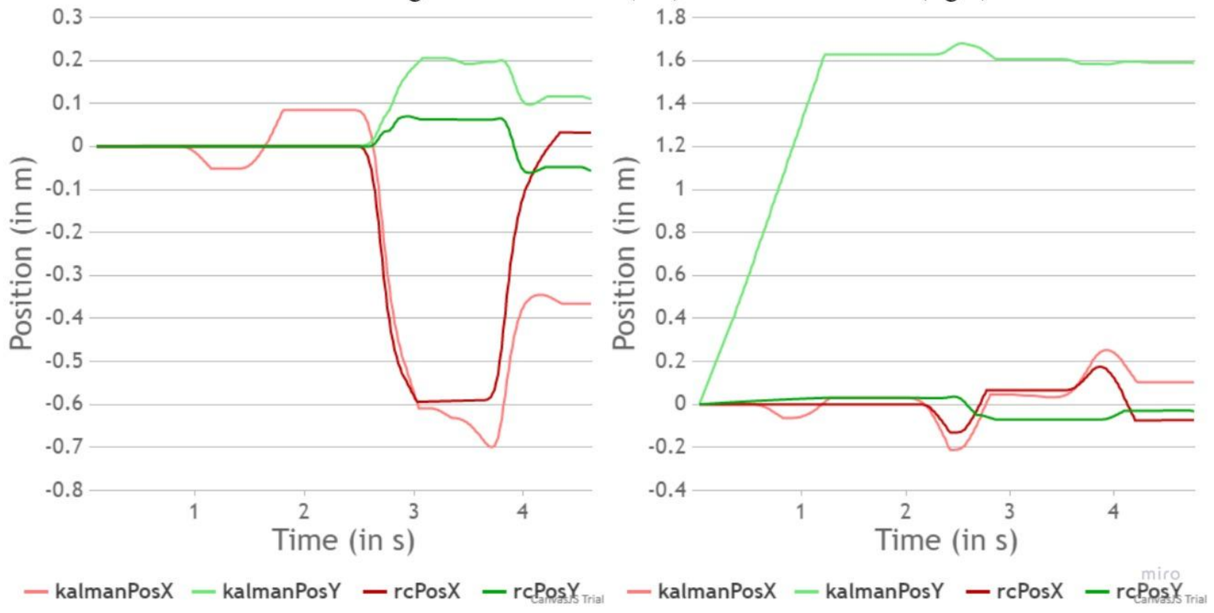
Figure 4.3.3.E : Representation of the position estimated via Algorithm 1 during the lateral movement throughout the first test (left) and the second test (right)



The data from the second and the third algorithms provided similar data. But on both tests, the first translation isn't accurate due to the movement being too small, so the author will also use the data with an origin at the start of the main movement. For the first translation, there is a good drift correction with a delta of approximately 0.1m but the Kalman algorithm estimates a small drift of approximately 0.15 m/s. For the second translation, the third algorithm detects a good movement a little smoother but the second algorithm detects a smaller movement causing a delta of 0.2m. For both the first movements, both algorithms detect a good drift correction for the Y axis but with a bit of inaccuracy +0.03m for the Kalman algorithm and -0.1m for the RC algorithm. The second translation on the Y axis is more accurate with an accuracy less than 0.05m.

For the second test, the second and third algorithms provide a really bad position estimation. Indeed, the two translations are incorrect due to a drift correction happening too early. It causes an early back and forth translation instead of a smoothstep translation. Movements detected in Y are pretty similar, with a good drift correction, despite the important drift at the start of the Kalman algorithm.

Figure 4.3.3.F : Representation of the position estimated via Algorithm 2 and 3 during the lateral movement throughout the first test (left) and the second test (right)



4.3.4 Horizontal Translation

The second movement is a horizontal translation. According to the real data, the first test is a translation of 1.1m on the Y axis at approximately 4.25m/s and a small translation of 0.4m on the X axis at 1m/s. The second test is a translation of 1.25m on the Y axis at approximately 2.25 m/s and a small translation of 0.4m on the X axis at 0.75 m/s.

Figure 4.3.4.A : Representation of the observed position of the device during the horizontal movement throughout the first test (left) and the second test (right)

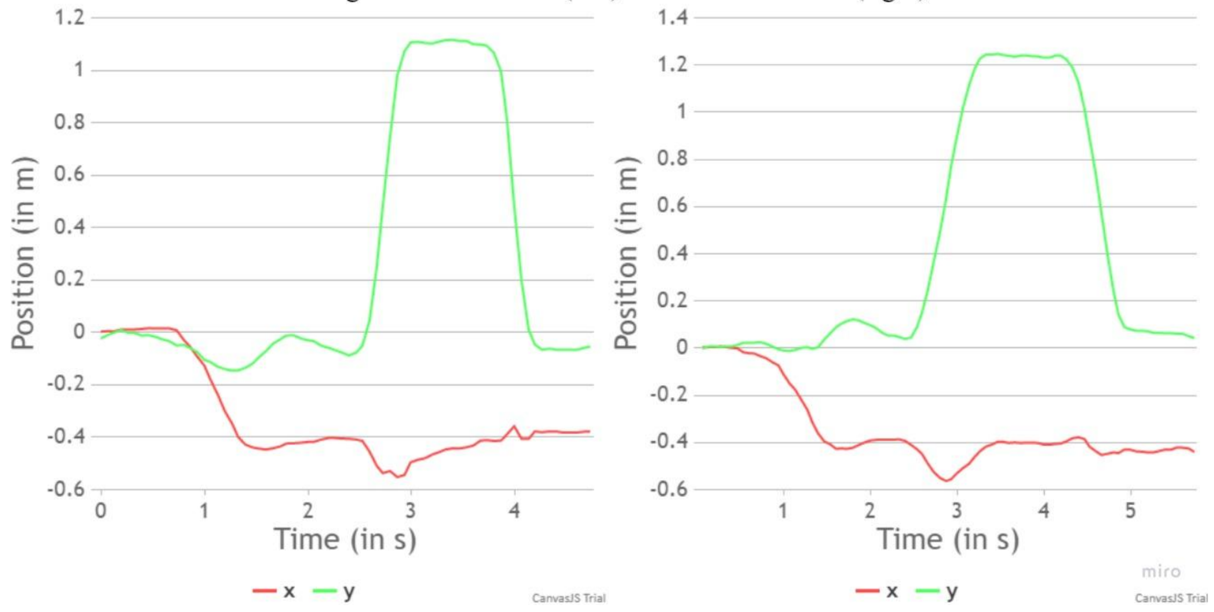
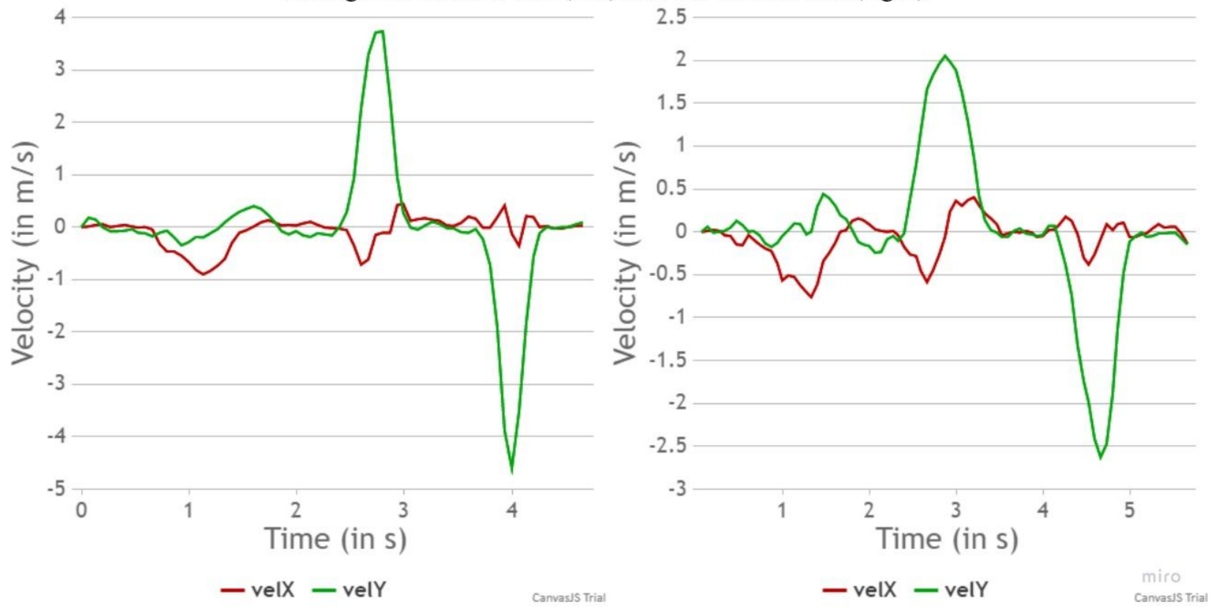
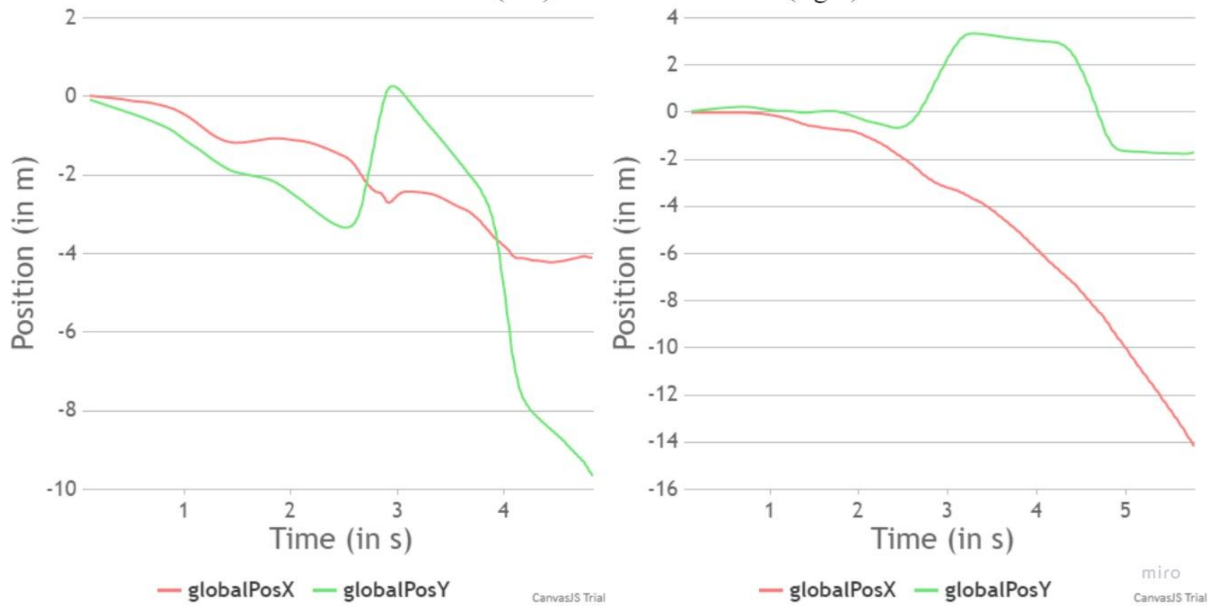


Figure 4.3.4.B : Representation of the observed velocity of the device during the horizontal movement throughout the first test (left) and the second test (right)



When analyzing the raw data of the first test, we can observe a drift on both axes but still recognize the movement. There is a drift of 1.6 m/s on Y and 1 m/s on X. For the second test, the raw data are already very accurate on the Y axis but with a translation 3 times bigger. However, the X axis provides an important drift of an average of 2.6 m/s that increases.

Figure 4.3.4.C : Representation of the estimated position during the horizontal movement throughout the first test (left) and the second test (right)



For the first test, the three algorithms provide similar results. Indeed, they are all really accurate on the Y detection during the movement with a delta less than 0.2m during the pause and at the end. For the X axis, the translation detected is less than 0.1m which is negligible. But, for the second test, the algorithms provide different results. The first algorithm is really accurate on the Y axis with a delta at the pause of 1.5m and a delta at the end less than 0.01m. For the X axis, the translation detected is less than 0.2m which is negligible. The second and third algorithm provides similar bad results. Indeed, with a drift correction happening too early, both translations provide false data. The X axis can also be neglected.

Figure 4.3.4.D : Representation of the position estimated via Algorithm 1 during the horizontal movement throughout the first test (left) and the second test (right)

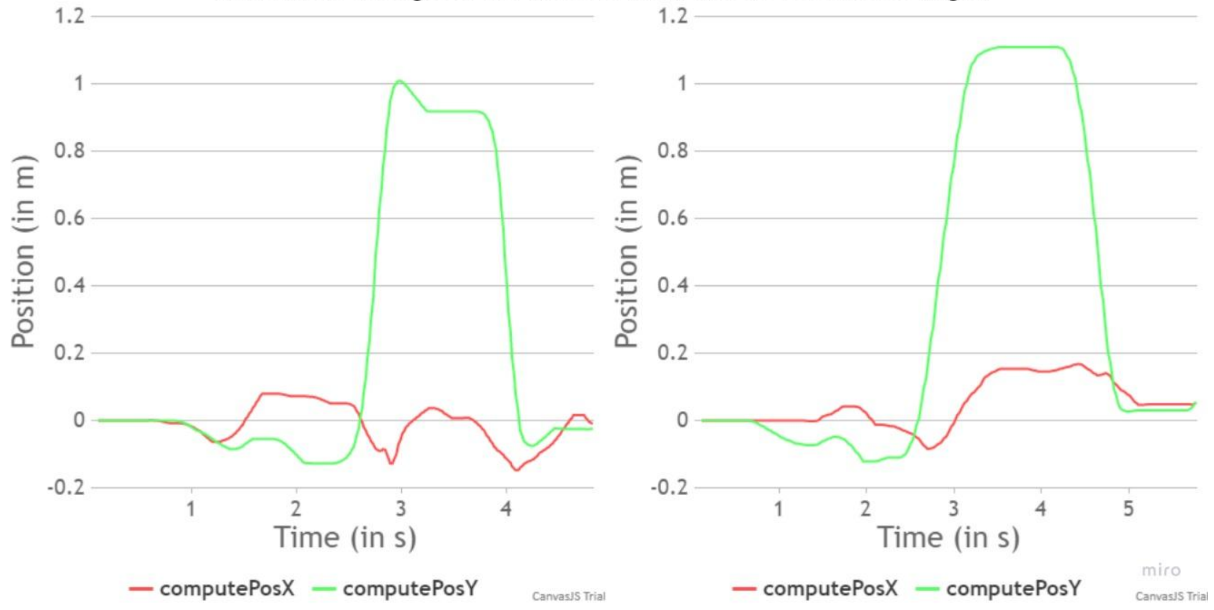
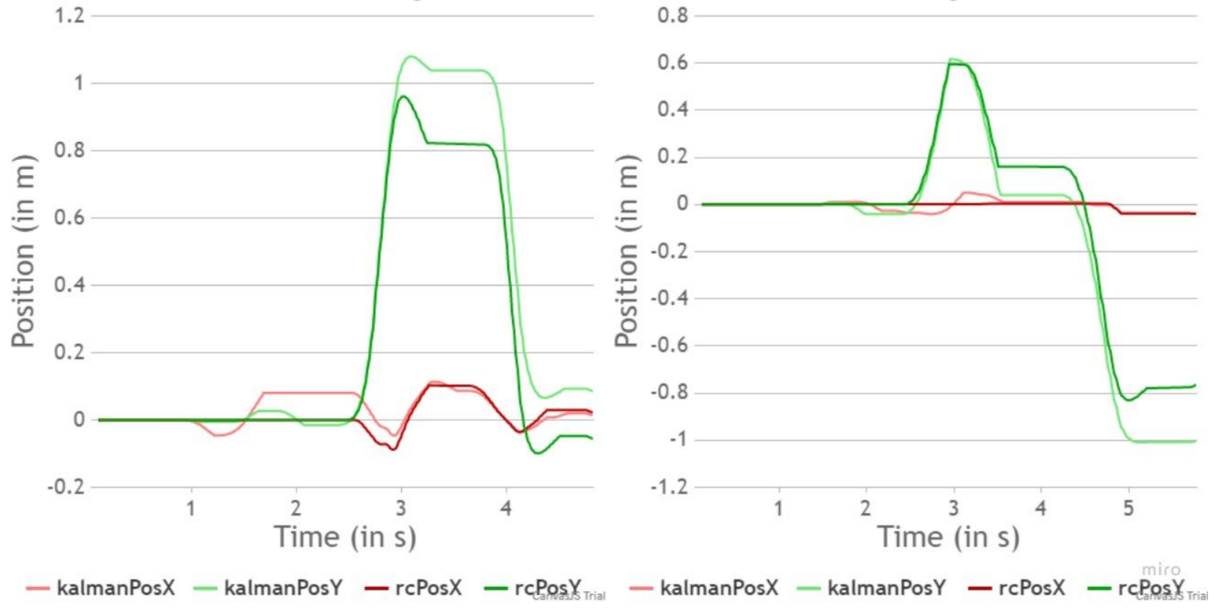


Figure 4.3.4.E : Representation of the position estimated via Algorithm 2 and 3 during the horizontal movement throughout the first test (left) and the second test (right)



4.3.5 Horizontal Arc Translation

The third movement is a horizontal arc movement. According to the real data, the first test is a translation of 1.15m on the Y axis at approximately 6.5m/s and a back-forth translation during the movement of 0.5m on the axis X at approximately 2.5m/s. The second test is a translation of 1.2m on the Y axis at approximately 3.4m/s and a back-forth translation during the movement of 0.5m on the axis X at approximately 1.5m/s. In the two cases, there is a first translation of 0.4m on the axis X at approximately 1m/s which is negligible because done for the placement.

Figure 4.3.5.A : Representation of the observed position of the device during the horizontal arc movement throughout the first test (left) and the second test (right)

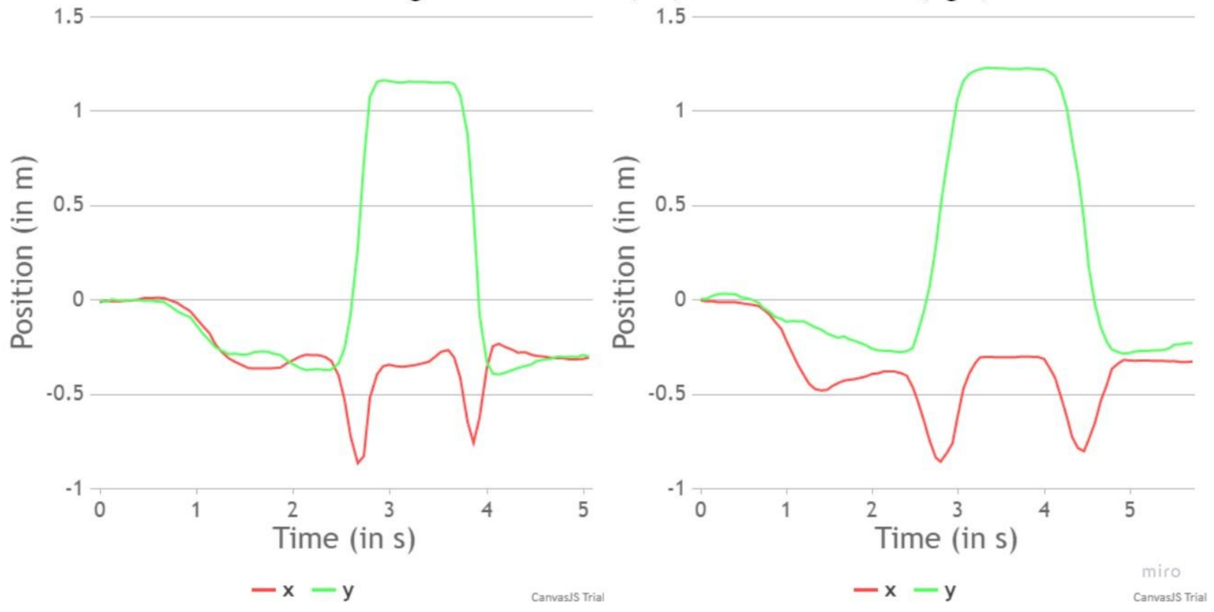
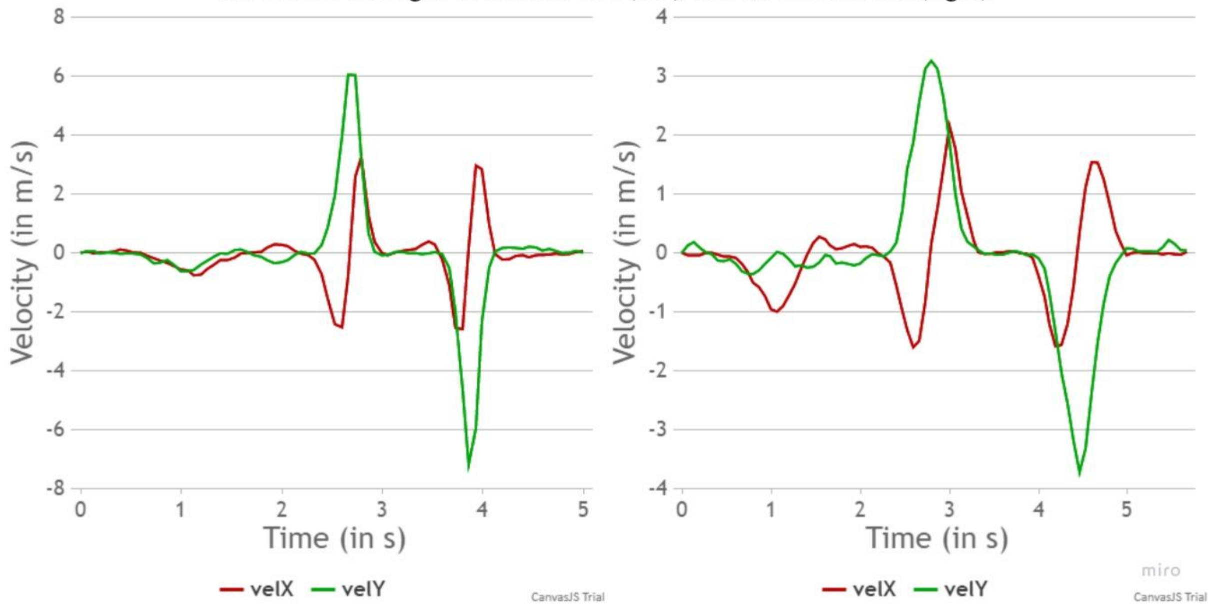
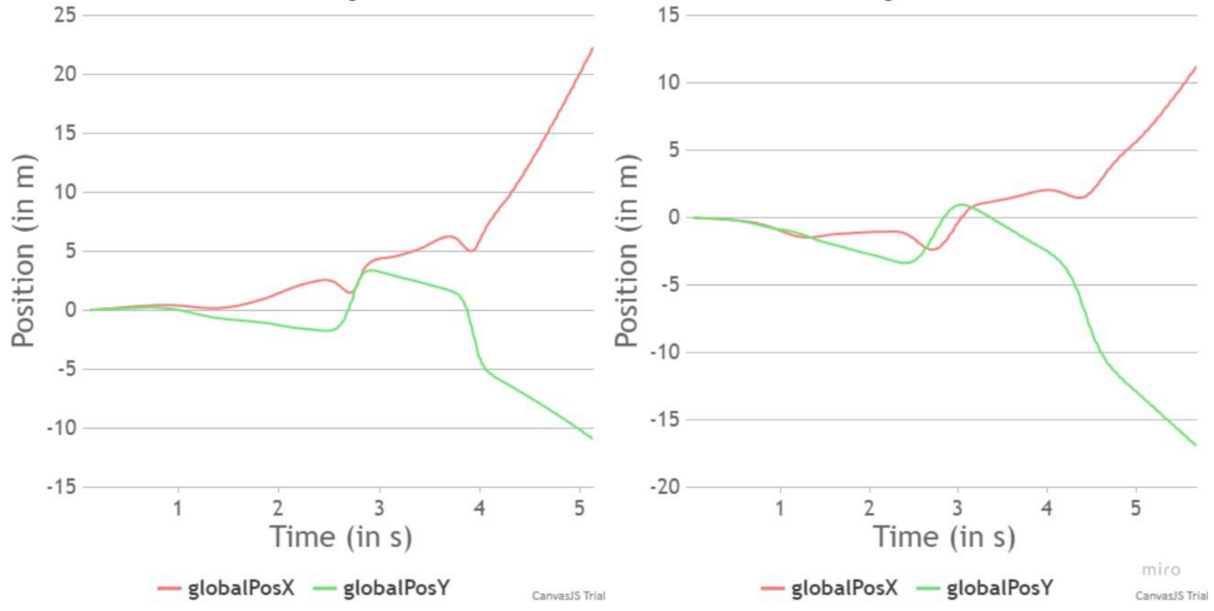


Figure 4.3.5.B : Representation of the observed velocity of the device during the horizontal arc movement throughout the first test (left) and the second test (right)



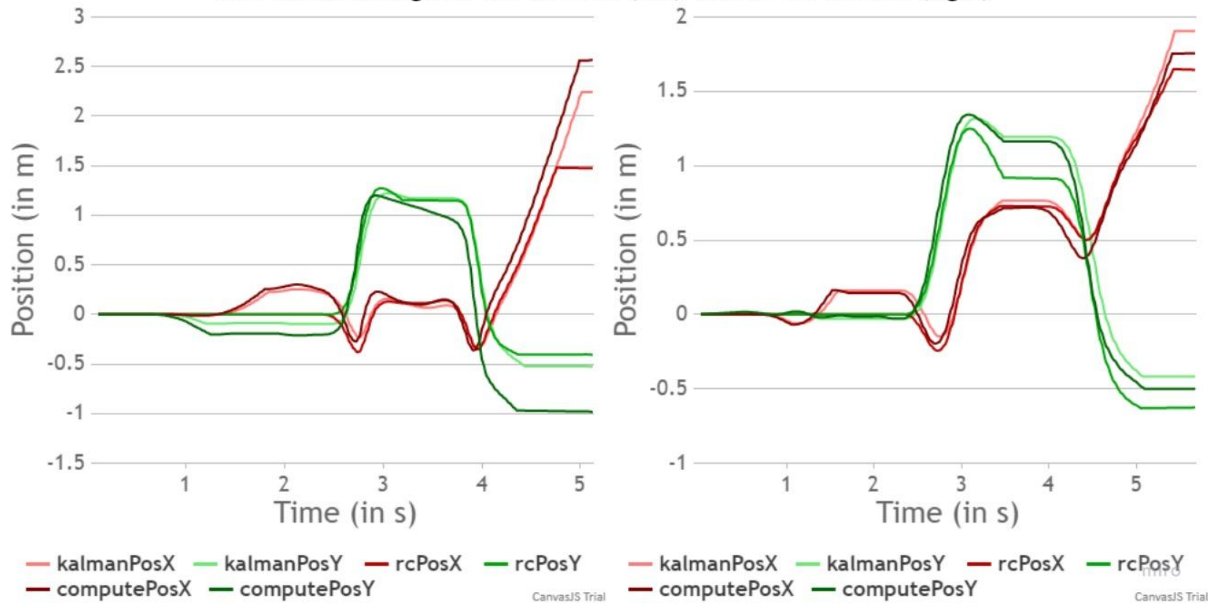
For both tests, the raw data provides drifting values on both axes but the movement is still recognizable. All the drifts are pretty similar with an average of 1.5m/s, all increasing.

Figure 4.3.5.C : Representation of the estimated position during the horizontal arc movement throughout the first test (left) and the second test (right)



For both tests, the three algorithms provide similar results. For the first test, the movement on the Y axis is very accurate with a good drift correction at the end of the movement causing a delta less than 0.1m with a drift during the pause of 0.34m/s for the algorithm with simple smoothness. On the X axis, the estimation of the start of the translation is inaccurate due to a too early drift correction. But the first movement is pretty accurate with a good drift correction. On the contrary, the second translation causes an important drift of 3 m/s.
For the second test, on the Y axis, the algorithms provide a good drift correction at the end of the movement causing a small delta of 0.2m at the pause. On the second translation, the drift at the end causes a delta of approximately 0.25m. On the X axis, the estimation of the start translation is also inaccurate due to a too early drift correction. The first movement is less accurate than the previous test with a drift at the end causing a delta of 0.7m. Like the previous tests, an important drift appears during the second movement of 1.5 m/s.

Figure 4.3.5.D : Representation of the position estimated via Algorithm 1, 2 and 3 during the horizontal movement throughout the first test (left) and the second test (right)



4.4 Review of the implementation

The program has been organized around a central script called CalculationFarm. This script stores every data of the current frames and calls every algorithm at every frame. Every algorithm is separate on a different script and sends the processed value to the CalculationFarm. The data are obtained by the OutputManager which saves each frame of each data and outputs them on a JSON file.

This organisation allows the program to easily create new algorithms and test multiple parameters. Graphs on Unity are also being developed in order to visualize directly the results as shown in Figure 4.2.3.B

5. Results analysis

The result can then be analyzed along with two main parts. The first part is how the algorithms reach the goal that is the precision during the movement and the precision at the end of the movement.
The second part is which parameters influence the results. Therefore, three parameters have been identified : the algorithms, the movements and the speeds.

5.1 How it reaches the goal

5.1.1 Precision during movement

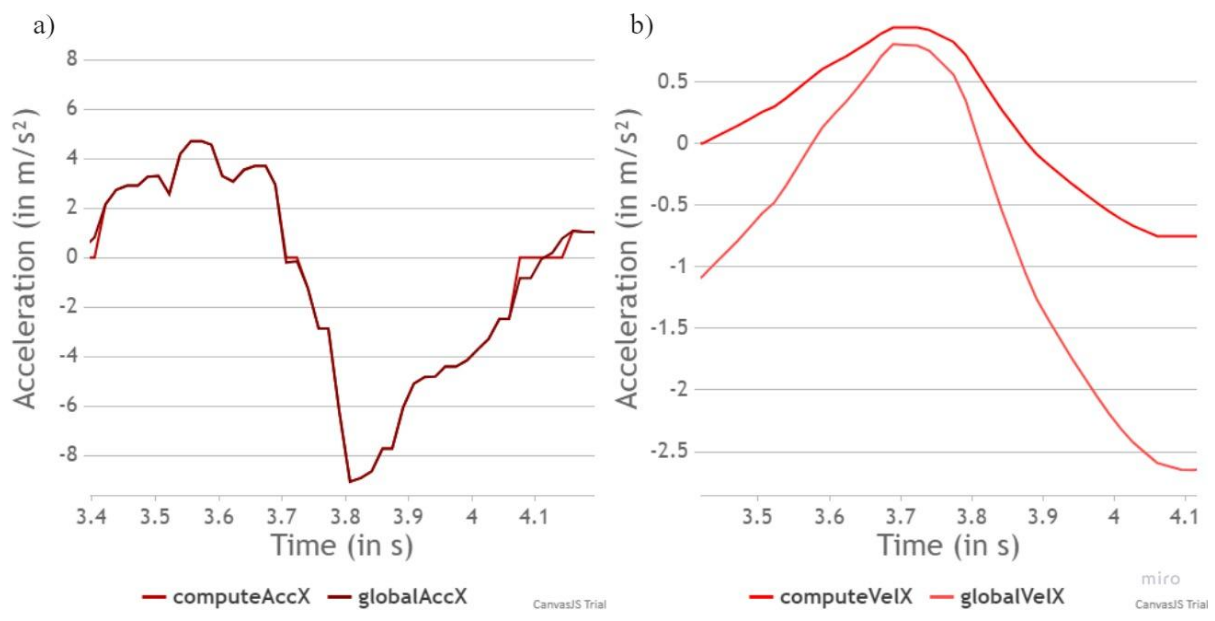
The precision during the movements has been separated into two phases. The first phase is to evaluate if the movement is recognizable. Once the movement is recognizable, it is interesting to evaluate the accuracy of the position during the movement of the device.

5.1.1.1 Movement Recognition

The first step is to define what is a recognizable movement. According to the Cambridge dictionary, the recognition is “a situation in which people know what something is when they see” (‘recognition’, 2021). In our case, the movement is recognizable if the movement can be identified during the estimation. For example, in Figure 4.3.3.C.a, it is possible to identify, despite the drift, the general shape of the movement. Therefore, it is possible to identify the movements of the main axis in every raw data. This observation implies that, despite the noise, the general movements of the smartphone are clearly recorded by the accelerometer.
On the contrary, the movements of the secondary axis aren’t recognizable automatically. Indeed, looking at Figure 4.3.3.C.b, the movement of the Y axis isn’t recognizable. Looking at Figure 4.3.5.C.a, it is also observable that the first translation before the placement isn’t recognizable through the drift. It means that, for small movements, the noise is too important and outshines the real movements.
Even if in a general state, the positions estimated by the algorithms are very recognizable, it happens that algorithms make the data unreadable.
Indeed, looking at Figure 4.3.3.F.b and Figure 4.3.4.E.b, the movement is totally unrecognizable. Indeed, the drift correction totally cuts the movement. It causes an early back and forth translation instead of a smoothstep translation.

Therefore, algorithms can totally modify a movement due to a too early drift correction.
Furthermore, the movement can also be unrecognizable due to the raw data.
When looking at Figure 4.3.3.E.b, the second movement is totally different because it describes a back-forth. This inaccuracy is not due to the algorithm but to the raw data providing inaccurate data. Indeed, looking at Figure 5.1.1.1.A, the algorithm doesn't modify the raw data, which means this inaccuracy is due to the raw data itself.

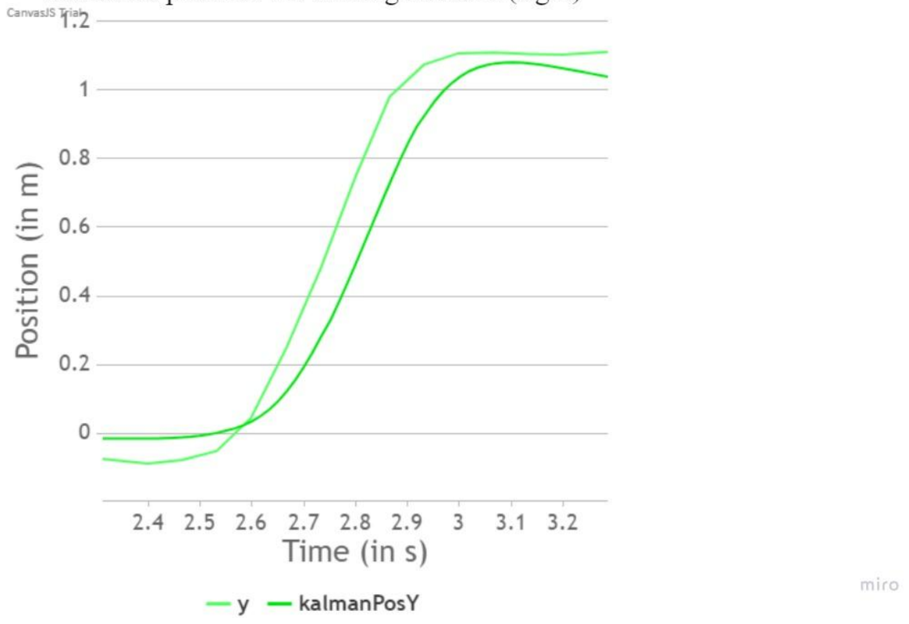
Figure 5.1.1.1.A : Representation of the estimated raw acceleration (left) and the raw velocity (right), calculated via Algorithm 1 during the lateral movement.



5.1.1.2 Movement Accuracy

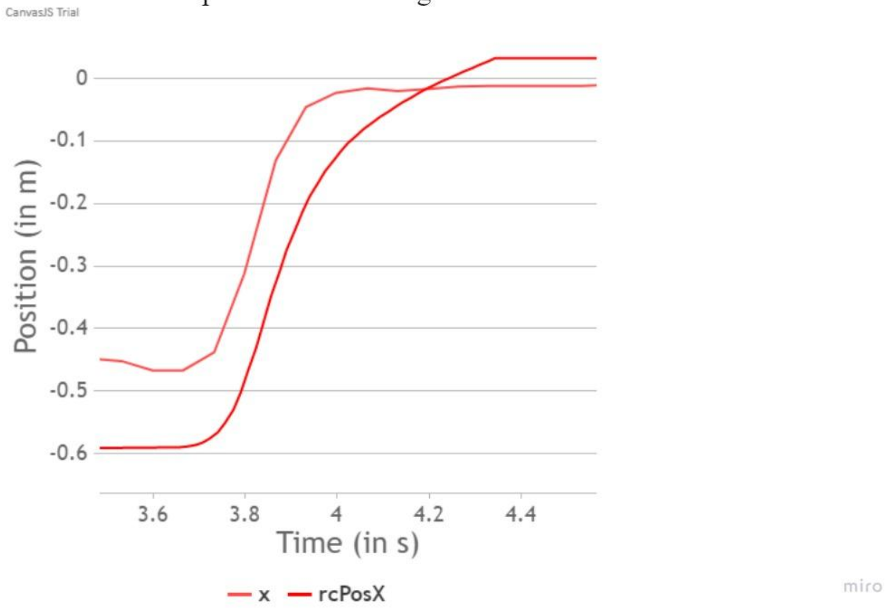
Once the movement has been recognized, it is interesting to analyze the accuracy of this one. Looking at the raw data, the drift prevents the analysis of the movement's accuracy. And even when there is a small drift, such as in Figure 4.3.4.C.b, the movement is totally inaccurate because it is three times bigger than the real movement.

Figure 5.1.1.2.A : Representation, during the first horizontal movement, of the real position and the estimated position via the Algorithm 2 (right)



Examining the algorithm, the movements are more accurate. Indeed, when looking at Figure 5.1.1.2.A, the movements are very accurate with a delta during the movement less than 0.1m. However, the algorithms don’t always provide an accurate movement. Indeed, a lot of movements are a bit too big such as in Figure 4.3.3.F.a or a bit too small such as in Figure 4.3.4.E.a. Furthermore, some estimated movements describe a different movement. Looking at Figure 5.1.1.2.B, the movement is longer and smoother than the real movement. This is due to the smoothing of the low-pass filter.

Figure 5.1.1.2.B : Representation, during the first lateral movement, of the real position and the estimated position via the Algorithm 3



5.1.2 Precision at the end of a movement

The precision at the end of a movement can also be described in two phases. The first phase is to check if the position at the end of the movement is static and doesn't suffer from drift. Once the position has been described as static, it is interesting to analyze the delta between the real position and the estimated position.

5.1.2.1 Static position without drift

The first thing to notice is that almost all raw data have a drift during movements. However, it is possible, if the noise of the raw data is not too important, that the raw data provide an estimation without drift. Indeed, when checking Figure 4.3.4.C.b, almost no drift is visible.
Even if in most of the cases, the drift correction provided by the algorithm is enough to detect a static position, sometimes it isn’t enough. For example in Figure 4.3.3.E.a, the compute algorithm provides data with drift because the raw acceleration received includes too much noise to be corrected by a simple smooth algorithm. Another important drift is visible despite the algorithm in Figure 4.3.5.D. Indeed, at the end of the second back-forth on the X axis, no algorithm manages to correct the drift. This important drift is certainly due to an end movement not stable enough to allow the drift correction to work properly.

5.1.2.2 End position accuracy

Once the position has been correctly estimated as static, it is interesting to analyze the accuracy of this end position. This accuracy can be analyzed by the delta between the real end position and the estimated end position. As seen in the chapter 4.3, the delta between the real and the estimated position varies between 0.1m and 0.2m when the drift has been corrected, such as in Figure 4.3.4.E.a, between 0.4m and 0.9m when the drift has been badly corrected, such as in Figure 4.3.3.E.b, and a drift more than 1m when the drift hasn’t been corrected, such as Figure 4.3.3.E.a or Figure 4.3.5.D. Furthermore, the delta between the real and the estimated position isn’t always in the same way. Indeed, looking at Figure 4.3.3.F.a, the delta is in the direction of the movement inducing a bigger movement and, looking at Figure 4.3.4.E.a, the delta is in the opposite direction of the movement, inducing a smaller movement.

5.2 What influence the results

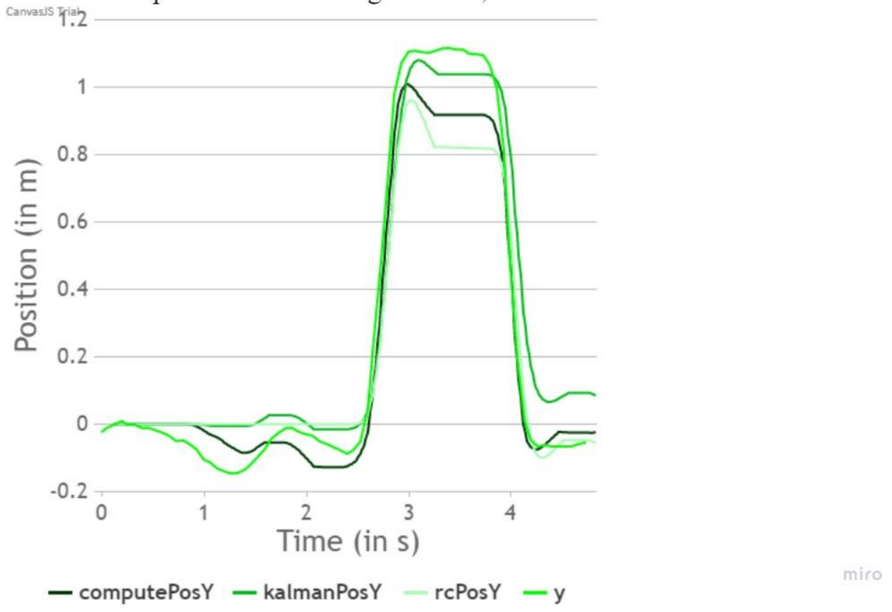
Once we analyze the accuracy of the algorithm, it is interesting to analyze which factors influence this difference of accuracy.

5.2.1 Algorithm

The first thing influencing the estimated position is the algorithm used. Indeed, with only the raw data, it has been observed that the estimation provides a drifting position. However, despite the drift, the movement can still be identified during the drift.
Therefore, the author has chosen to put three algorithms in parallel in order to be able to compare their efficiency. These three algorithms use the same parameter for all tests.
The first observation is the similarity of the results between the three algorithms. Looking either at Figure 4.3.4.D and Figure 4.3.4.E or at Figure 4.3.5.D, the three algorithms provide similar position estimations. Furthermore, the second and the third algorithm provide even similar results, such as on Figure 4.3.4.E.b. However, it is possible to analyze multiple differences between these algorithms.
The first difference is the capacity to remove drift. During the first test on lateral movement, after the first movement, the data provided by the accelerometer is composed of too much noise. Looking at Figure 4.3.3.E.a, the first algorithm provides no drift correction. However, looking at Figure 4.3.3.F, the second algorithm provides a smoothed drift correction less than 0.75 m/s. Finally, the third algorithm provides a good drift correction with a static position. This difference can be explained by the strength of the smoothing algorithm. The first algorithm doesn't apply an important smooth. Therefore, its estimation doesn’t reduce the noise enough. The Kalman algorithm applies a medium smooth and its estimation reduces the noise, but not enough. Finally, the high-pass filter applies an important smooth allowing the total removal of the noise.
The second difference is the application of an early drift correction. During the second test on the lateral movement, the first movement is pretty accurate for the first algorithm. However, for the second and the third algorithm, an early drift correction totally modifies the movement. This is also caused by the strength of the smoothing algorithm. The first algorithm removed just enough noise to keep the general shape of the movement. However, the second and the

third noise apply too much smoothing triggering the drift correction at the start of the slowdown.

Figure 5.2.1.A : Representation, during the first horizontal movement, of the real position and the estimated position via the Algorithm 1, 2 and 3



Finally, the smoothing also changes the precision of the movement. While looking at Figure 5.2.1.A, the second algorithm provides a movement similar to the real movement with a delta less than 0.05m. The first algorithm provides a smoother movement with a delta of 0.1m and the third movement provides an even smoother movement with a delta of 0.2m. Therefore, the smooth strength also influences the accuracy of the movement.
Even if the three algorithms have different ways of applying smoothing, it can be observed that the smoothing strength influences the result more than the way the smoothing is applied.

5.2.2 Movement

The movement also influences the accuracy of the estimated data.
The first thing to observe is the difference between the main movements. First, it can be observed that there is a difference of accuracy between the lateral and horizontal movement. However, this difference can be due to multiple causes such as a small movement, the smartphone being more stable along its length than its width or the accelerometer being more accurate along the Y axis than the X axis. Furthermore, it can be observed that a rotating movement doesn't influence the accuracy of the test. Indeed, looking at Figure 4.1.1.A, it can be observed a pretty accurate rotation between the observed and the estimated orientation.

A second movement that influences the data is the small movements. Looking at Figure 4.3.4.D.b, it can be observed that, at the start, the Y axis, which moves during a small movement, is inaccurate and totally unrecognizable. This inaccuracy in the small movement is visible for almost every secondary axis but not only. Indeed, this inaccuracy in small movements is also visible during the first small translation. Looking at Figure 4.3.3.E.a, the first translation has been totally removed from the estimated position.
Finally, the stability of the movement greatly influences the accuracy of the estimation. Indeed, looking at Figure 4.3.3.E.a, the drift is caused by too much noise due to a static position not stable enough. Furthermore, looking at Figure 4.3.5.D, the drift at the end is also caused by not some instability during the static position.

5.2.3 Velocity

A third factor that influences the accuracy of the estimation is the velocity of the movement.
For each movement, the author has chosen to move the device at different velocities to see if the velocity influences the estimation. And multiple times, the estimations between the first test, faster, and the second test, slower, are really different.
The first difference is the apparition of the early drift correction. In Figures 4.3.3.F and 4.3.4.E, the early drift correction appears only on the second test. Indeed, a slower test also applies a longer movement, approximately 0.5s for the first test and 1s for the second test. While, during the first test the gap between the acceleration and the deceleration is less than 0.2s, during the second test, it is approximately 0.4s, allowing the drift correction to detect a static movement.
The second difference is the disappearance of slow movements. Indeed, just like the small movements, the slow movements are also ignored by the estimation. For the lateral and horizontal movement, on the secondary axis, all the velocities lower than 0.5m/s have been removed by the smooth algorithm. Furthermore, the slow translation, less than 1m/s, at the start of the lateral movement has been modified by an early drift correction because it was too slow.

5.3 Summary of findings

By doing these analyses, some important features can be observed.
First of all, the movement is almost always recognizable even in the raw data. However, there are two cases in which it is unrecognizable. During small or slow movements, such as for the secondary axis or during the first translation, the movement is totally modified by the algorithm. Then, the movement is also unrecognizable when there is an early drift correction.
The second point is that, even if the raw data drift in almost every test, with the algorithm, the drift correction is correctly applied almost every time with a drift less than 0.5m. However, the drift can still appear when there is too much noise or instability.
The third point is the influence of the algorithm on the accuracy of the estimation. Indeed, a stronger smooth provides more drift correction but can also create too much drift correction leading to an early drift correction. Furthermore, a stronger smooth induces a less accurate movement.
The fourth point is the influence of the movement on the accuracy of the estimation. It can be observed some difference of accuracy along the movement but there is no difference induced by the rotation during the movement. The movement influences the accuracy if it's too small or too unstable.
Finally, the velocity also influenced the accuracy. Indeed, a slow movement implies less accuracy because it can create an early drift correction or it can be totally smoothed.

Conclusion

In this thesis, the objective was to improve the position estimation of a smartphone using inertial sensors during fast movements.

First of all, two main issues of inertial tracking have been identified : the multiple integrations and the lack of external anchors. However, some solutions have been found such as the smoothing algorithms or the drift correction algorithms.

Then, the analysis of different projects allowed the highlight of different smoothing algorithms and in which case they are used. These analyses highlighted the fact that the smoothing algorithm to use depends mainly on the movement that the device has to detect.

Therefore, six tests have been done to be able to evaluate the precision of the estimation during fast movements with a static start and a static end.

Finally, the analysis of the tests highlighted the disparity of the results, detecting both good and bad results. The analysis of the results also highlighted the influence of the algorithm, the movement and the speed on the disparity of the results.

Indeed, some good points have been identified, such as the really good precision of some movement and the good drift correction for static points. Furthermore, the tests allow to highlight the efficiency of the algorithms during diverse movements such as straight translation or rotating arc translations.

However, diverse problems have also been detected.

First of all, some algorithms apply no drift correction due to too much noise in static position. This can be corrected by increasing the drift correction threshold or by increasing the smooth parameter.

Another problem is that the estimation of the movement is totally modified by a too early drift correction. This problem can be corrected by decreasing the drift correction threshold or decreasing the smooth parameter.

Finally, the last main problem is the avoidance of small and slow movements. This can be modified by greatly decreasing the smoothing parameter and the drift correction threshold.

Therefore, due to all these defects, the algorithm is not usable in a game. Indeed, it needs to be a minimum universal to be usable by a diversity of people. But the current algorithm needs movements with an important speed and to be really static between movements.

Going further

As seen before, some changes can be done to correct the algorithm. But, these problems need opposite solutions, one needs to increase the drift correction when the other one needs to decrease the drift correction. Therefore, a good way to go further is to apply adaptive parameters. Indeed, current parameters are constant for the whole movement. A good way to correct them is to apply a different drift correction between the movement and the static position. To separate this parameter, it is possible to apply pattern recognition like it has been done for step recognition. Another way to create adaptive parameters is to apply machine learning. This technology eases the recognition of patterns and corrects the parameters according to the pattern recognized.
Finally, this research was made with the current technologies present in smartphones. With the evolution of technologies, better sensors in smartphones can greatly facilitate position estimation algorithms.

If some like to take a deeper look into the code, the public repository of the project is accessible at : https://github.com/FloreauLuca/IMU-MotionTracking

Bibliography

Allan, A. (2011). Basic Sensors in iOS: Programming the Accelerometer, Gyroscope, and More (1st edition). O'Reilly Media.
AlterGeo—Global location technology provider. (2021). https://altergeo.ru/?mode=about

Arduino—Introduction. (n.d.). Retrieved 3 September 2021, from https://www.arduino.cc/en/guide/introduction?setlang=en

Bernstein, J. (2003, February 1). An Overview of MEMS Inertial Sensing Technology | FierceElectronics. https://www.fierceelectronics.com/components/overview-mems-inertial-sensing-technology

Bias (statistics). (2021). In Wikipedia. https://en.wikipedia.org/w/index.php?title=Bias_(statistics)&oldid=1032863292

Cooper, S., & Durrant-Whyte, H. (1994). A Kalman filter model for GPS navigation of land vehicles. Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94), 1, 157–163 vol.1. https://doi.org/10.1109/IROS.1994.407396

Dead reckoning. (2021). In Lexico Dictionaries | English. https://www.lexico.com/definition/dead_reckoning

Dixon-Warren, St. J. (2010, December 23). Motion sensing in the iPhone 4: MEMS accelerometer. MEMS Journal -- The Largest MEMS Publication in the World. https://www.memsjournal.com/2010/12/motion-sensing-in-the-iphone-4-mems-accelerometer.html

EI. (2019, July 27). IMU Sensors: Everything You Need To Know! Embedded Inventor. https://embeddedinventor.com/what-is-an-imu-sensor-a-complete-guide-for-beginners/

Filter (signal processing). (2021). In Wikipedia. https://en.wikipedia.org/w/index.php?title=Filter_(signal_processing)&oldid=1039176584

Foundations and Trends in Signal Processing—Impact Factor, Overall Ranking, Rating, h-index, Call For Paper, Publisher, ISSN, Scientific Journal Ranking (SJR), Abbreviation, other Important Details | Resurchify. (2021, June 24). https://www.resurchify.com/impact/details/19600166419

Google TechTalks. (2010). Sensor Fusion on Android Devices: A Revolution in Motion Processing. https://www.youtube.com/watch?v=C7JQ7Rpwn2k

GSM Association. (2021). GSMA | The Mobile Economy 2021. https://www.gsma.com/mobileeconomy/

Gyroscopes—MEMS and Sensors—STMicroelectronics. (2021). https://www.st.com/en/mems-and-sensors/gyroscopes.html

High-pass filter. (2021). In Wikipedia. https://en.wikipedia.org/w/index.php?title=High-pass_filter&oldid=1036752882

Indoor positioning system. (2021). In Wikipedia. https://en.wikipedia.org/w/index.php?title=Indoor_positioning_system&oldid=1027084903

Jurkovich, T. (2021, February 11). 15 Best Games That Use Motion Sensor Controls, Ranked. TheGamer. https://www.thegamer.com/games-best-motion-sensor-controls-ranked/

Kinect. (2021). In Wikipedia. https://en.wikipedia.org/w/index.php?title=Kinect&oldid=1034468707

Kirk, A. G., O'Brien, J. F., & Forsyth, D. A. (2005). Skeletal parameter estimation from optical motion capture data. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2, 782–788 vol. 2. https://doi.org/10.1109/CVPR.2005.326

Kok, M., Hol, J. D., & Schön, T. B. (2017). Using Inertial Sensors for Position and Orientation Estimation. Foundations and Trends® in Signal Processing, 11(1–2), 1–153. https://doi.org/10.1561/2000000094

Leporcq, J. (2018). Position Estimation Using IMU Without Tracking System. https://aaltodoc.aalto.fi:443/handle/123456789/32426

Lin, J., Zhang, R., Chen, J., Zeng, L., Jiang, L., McGrath, S., & Yang, J. (2018). Using Hybrid Sensoring Method for Motion Capture in Volleyball Techniques Training. 2018 12th International Conference on Sensing Technology (ICST), 217–222. https://doi.org/10.1109/ICSensT.2018.8603555

Lindner, T., Fritsch, L., Plank, K., & Rannenberg, K. (2004). Exploitation of Public and Private WiFi Coverage for New Business Models. In W. Lamersdorf, V. Tschammer, & S. Amarger (Eds.), Building the E-Service Society (pp. 131–148). Springer US. https://doi.org/10.1007/1-4020-8155-3_8

Liu, X., Liu, J., Wang, W., He, Y., & Zhang, X. (2018). Discovering and understanding android sensor usage behaviors with data flow analysis. World Wide Web, 21(1), 105–126. https://doi.org/10.1007/s11280-017-0446-0

Llorach, G., Evans, A., Agenjo, J., & Blat, J. (2014). Position estimation with a low-cost inertial measurement unit. 2014 9th Iberian Conference on Information Systems and Technologies (CISTI), 1–4. https://doi.org/10.1109/CISTI.2014.6877049

Low-pass filter. (2021). In Wikipedia. https://en.wikipedia.org/w/index.php?title=Low-pass_filter&oldid=1039066300
Luinge, H. (2002). Inertial Sensing of Human Movement (Twente University Press). http://edge.rit.edu/edge/P10010/public/PDF/mvn_Inertial_Sensing_of_Human_Movement_Thesis_Luinge.pdf

Madgwick, S. (2013a, September 2). Gait tracking with x-IMU – x-io Technologies. https://x-io.co.uk/gait-tracking-with-x-imu/

Madgwick, S. (2013b, November 3). Oscillatory motion tracking with x-IMU – x-io Technologies. https://x-io.co.uk/oscillatory-motion-tracking-with-x-imu/

Markovska, M., & Svensson, R. (2019). Evaluation of drift correction strategies for an inertial based dairy cow positioning system.: A study on tracking the position of dairy cows using a foot mounted IMU with drift correction from ZUPT or sparse RFID locations.
http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-263045

Milette, G., & Stroud, A. (2012). Professional Android Sensor Programming (1st edition). Wrox.
MobiCom 2021: The 27th Annual International Conference on Mobile Computing and Networking. (2021).
https://www.sigmobile.org/mobicom/2021/index.html

Nield, D. (2020, June 29). All the Sensors in Your Smartphone, and How They Work. Gizmodo.
https://gizmodo.com/all-the-sensors-in-your-smartphone-and-how-they-work-1797121002

Oculus | Casques, jeux et équipement VR. (2021). https://www.oculus.com/

O'Haver, T. (2021, July). Intro. To Signal Processing:Smoothing.
https://terpconnect.umd.edu/~toh/spectrum/Smoothing.html

Optical Tracking—An overview | ScienceDirect Topics. (n.d.). Retrieved 3 September 2021, from
https://www.sciencedirect.com/topics/computer-science/optical-tracking

Petric, T., Gams, A., Ude, A., & Zlajpah, L. (2010). Real-time 3D marker tracking with a WIIMOTE stereo vision system: Application to robotic throwing. 357–362. https://doi.org/10.1109/RAAD.2010.5524560

Recognition. (2021). In Cambridge Dictionary.
https://dictionary.cambridge.org/fr/dictionnaire/anglais/recognition

Satellite Navigation—GPS - How It Works. (2021, August 12). Federal Aviation Administration. https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps/howitworks/

Shen, S., Gowda, M., & Roy Choudhury, R. (2018). Closing the Gaps in Inertial Motion Tracking. Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, 429–444. https://doi.org/10.1145/3241539.3241582

Silicon Sensing | MEMS Gyroscopes. (2017). https://www.siliconsensing.com/technology/mems-gyroscopes/

Smoothing. (2020). In Wikipedia. https://en.wikipedia.org/w/index.php?title=Smoothing&oldid=977829507

Statista. (2021). Global VR gaming market size 2024. Statista. https://www.statista.com/statistics/499714/global-virtual-reality-gaming-sales-revenue/

Suprem, A., Deep, V., & Elarabi, T. (2017). Orientation and Displacement Detection for Smartphone Device Based IMUs. IEEE Access, 5, 987–997. https://doi.org/10.1109/ACCESS.2016.2631000

Taillet, R., Villain, L., & Febvre, P. (2018). Dictionnaire de physique. De Boeck Superieur. https://books.google.fr/books?id=pjlFDwAAQBAJ&pg=PP1#v=onepage&q&f=false

Ubilla, M., & Cadiz, R. (2010). Head Tracking For 3d Audio Using The Nintendo Wii Remote.
Understanding the accelerometer noise specification | Wilcoxon Sensing Technologies. (n.d.). Retrieved 3 September 2021, from https://wilcoxon.com/resources/technical-notes/understanding-the-accelerometer-noise-specification_tn33/

Unity QA - LTS Releases. (2021). Unity.
https://unity3d.com/unity/qa/lts-releases

Unity—Scripting API: Input. (2020). Unity Technologies.
https://docs.unity3d.com/ScriptReference/Input.html

Unity—Scripting API: Quaternion. (2020). Unity Technologies.
https://docs.unity3d.com/ScriptReference/Quaternion.html

Virtual reality. (2021). In Wikipedia.
https://en.wikipedia.org/w/index.php?title=Virtual_reality&oldid=1041510556

VR positional tracking. (2021). In Wikipedia.
https://en.wikipedia.org/w/index.php?title=VR_positional_tracking&oldid=1034630530

Wii. (2021). In Wikipedia.
https://en.wikipedia.org/w/index.php?title=Wii&oldid=1039663481

Woodford, C. (2020, October 8). How accelerometers work | Types of accelerometers. Explain That Stuff.
http://www.explainthatstuff.com/accelerometers.html

Zanetti, R., & D'Souza, C. (2020). Inertial Navigation. In J. Baillieul & T. Samad (Eds.), Encyclopedia of Systems and Control (pp. 1–7). Springer.
https://doi.org/10.1007/978-1-4471-5102-9_100036-1

List of Appendix

List of Figures

List of Abbreviations

VR	Virtual Reality
6-DoF	6 Directions of Freedom
GPS	Global Positioning System
IMU	Inertial Measurement Unit
MEMS	Micro Electro Mechanical System
EKF	Extended Kalman Filter

List of Algorithms