國 立 交 通 大 學

電控工程研究所

碩 士 論 文

使用加速規之慣性滑鼠裝置訊號處理方法評估

Evaluations of Signal Processing Methods for an Inertial Mouse Device Using Accelerometers

研 究 生：　活　多　福

指導教授：　胡　竹　生　博士

中 華 民 國 九 十 九 年 六 月

使用加速規之慣性滑鼠裝置訊號處理方法評估

研究生：活　多　福　　　　　　指導教授：胡　竹　生　博士

國立交通大學

電控工程研究所碩士班

摘要

本論文提出了一個以三軸加速度計來取代普遍用於商業滑鼠裝置的光學感測器的新型慣性滑鼠裝置。本 論文所使用的慣性感測器為加速度計，其具有減少能量耗損、縮減整體產品大小以及減低整體產品成本的優點，另外因為此種新型的 慣性滑鼠裝置可以隔空使用，所以也增加了其可使用的範圍。

本論文以光學感應器及三軸加速度計來實現所提出的 慣性滑鼠裝置，因此能在相同環境及條件下比較兩種不同感應器（ 光學感應器及慣性感應器 ）的效果。本論文以不同的數學方法以及從加速度計所得到的訊號來估測滑鼠裝置的位移量。 在最初時，這些實現的演算法皆以個人電腦為核心並搭配慣性滑鼠裝置來做測試，最後最適合的演算法則以微控制器來實現，整個 裝置成為一個獨立的新型慣性滑鼠裝置。

實 驗結果顯示出，本論文所提出的以加速度計為基礎的最佳估測技術可 以做為新型的慣性滑鼠裝置，且應用在一般的電腦上可以順利完成大多數的工作。

Evaluations of Signal Processing Methods for an Inertial Mouse Device Using Accelerometers

Student： Rodolfo Gondim Lóssio Advisor： Prof. Jwu-Sheng Hu

Institute of Electrical and Control Engineering
National Chiao-Tung University

ABSTRACT

This work proposes and evaluates an inertial mouse device based on three-axis accelerometer to be a substitute of the optical sensor, which is commonly used in the majority of commercial mouse devices. The use of inertial sensors, such as accelerometers, will allow reducing power consumption, physical dimensions and final product cost, moreover will increase easy-of-use, since this kind of mouse could be also used in free space.

For this purpose, a prototype containing an optical sensor and 3-axis accelerometer is built. In this way, it is possible to compare the two sensors under the same environment and conditions. In a first moment, different mathematical approaches are tested to estimate the displacement based on the acceleration signal. Those approaches are processed under a computer application connected to the prototype. By the end, the most suitable algorithms are ported to the microcontroller embedded in the prototype.

The result of the experiments show that the best estimator techniques based on the accelerometers can be used as a mouse device to perform the majority of the tasks when interacting with a computer.

Acknowledgments

My first words will go to my family in Brazil, which will be written in Portuguese:

“Aos meus pais e irmãos, gostaria de agradeçe-los por todo o suporte que vocês me deram durante esses dois anos e pouco que estive no exterior estudando e trabalhando. Sem esse suporte, acredito que não conseguiria ir tão longe nos meus estudos e na minha vida profissional. Sinto muita falta de vocês e gostaria muito que vocês estivessem na minha cerimônia de graduação. Mas por causa da distância, tempo e dinheiro, esse desejo torna-se quase impossível. Mesmo assim, espero que no futuro vocês possam conhecer esse país que me conquistou e deu enorme oportunidades para meu crescimento profissional e acadêmico.”

For the people from Taiwan, at first I would like to thank my advisor, Prof. Hu, for accepting to orientate my research for these two years. It was a pleasure for me to have worked together with such a really competent and qualified professor.

Also, I would like to thank my labmates from X-Lab, who helped me a lot, specially with the bureaucratic stuff that concerns with the academic life. I made really good friends in here, and I hope we can keep in touch for a long time.

And for last, I would like to thank my girlfriend, Charlene, for helping me since my first month in Taiwan. It is not easy to stay far away from family. But with her help, I overcome this problem, and she is one of the main reasons I feel at home in Taiwan now.

Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 Motivation and Objective

In the last decades, many commercial mouse devices were developed with different technologies. The first generation of a commercial mouse device is so called mechanical mouse, which uses a single ball that can rotate in any direction. This ball is connected against to two rollers. One roller detects the forward-backward motion and other the left-right motion. The movement of these two rollers is detected by an encoder and an electrical signal is send to the computer. In the computer, a driver software in the operation system converts the signal into motion of the mouse cursor along X and Y axes on the screen [15]. The disadvantage of this kind of mouse is that it often requires maintenance, due to the moving parts that can easily accumulate dust and lint. Besides that, it does not perform well in slippery surfaces, requiring in most of the cases a mouse pad for better performance.

The second generation of mouse devices is so called optical mouse, which uses an optoelectronic sensor that takes successive pictures of the surface on which the mouse operates. The surface is illuminated by an LED or a laser diode. Changes between one frame and next are processed by the image processing part of the chip and translated into movement on the two axes using a block matching algorithm. Comparing this generation of mouse device with the previous one, it presents higher sensitivity and practically do not require any maintenance. However, most of the optical mouse do not work well in glossy and transparent surfaces and demand a higher average of power.

The next generation of mouse devices that is starting to appear in the market is called inertial mouse devices, which can use gyroscope or accelerometer sensors to detect movement for each axis supported. These two kinds of sensors consume quite less power than the optical sensors, also has a huge potential to cost less than optical sensors after mass production. Besides that, a new way to interact with computer systems will be allowed, since they do not require a surface to operate. Moreover, when using a wireless battery-powered mouse device, it will increase easy-of-use and due to the small consume of energy, it can be used during long period of time without recharging. Another benefit of using inertial sensors is their size. They are extremely small ICs that can easily be embedded in unusual objects, like a ring, a watch or glasses. Such objects that may be used as mouse devices, especially for handicap people.

In the market, it is already possible to find hybrid devices that use optical sensor and inertial sensor. The first one is only used in a 2D surface and the second one is used on fly. However, adding an optical sensor would increase the final product cost, the power consumption and the product size.

The objective of this thesis is to propose and evaluate an inertial device mouse based on a three-axis accelerometer that can substitute the optical sensor in a hybrid device, in this way, only inertial sensors will be embedded in the system, reducing the total power consumption, the production cost and size.

1.2 Survey of Previous Work

A patent [1] claiming an inertial mouse system based on accelerometers was filed in 1988, describing that such mouse would consume less power than optically based mouse, and offer increased sensitivity, reduced weight and increased easy-of-use. Since then, many accelerometer sensors were designed to be used for mouse applications **Error! Reference source not found.**. However, because the nature of estimating the displacement based on acceleration signals is extremely difficult, there is not such a mouse device yet in the market competing with the optically based mouse.

The biggest challenge is to apply accurate signal processing methods to integrate the acceleration signal. This integration can be as simple as the one proposed in [7], or as complicated as in [2], which uses Kalman Filter. Another way to estimate the displacement is to use pattern recognition algorithms, as the one proposed in **Error! Reference source not found.**, which uses Fuzzy-Neural Networks.

Most of the research papers used as survey for this thesis propose signal processing techniques to be applied with accelerometers for different applications than mouse devices, such as robot positioning [3], gesture recognition **Error! Reference source not found.**, static balancing control of humanoid robots [20], detection of small displacement for portable devices [18] and detection for human actions [19]. Accelerometers also are common designed as a sensor for inertial navigation systems [17], especially in mobile robot applications, as in [21] and [22].

The other few papers that use accelerometers for mouse device systems are based on tilt angle [23], which instead of performing translation movements, as used in optically based mouse, the user must rotate the device to move the cursor on screen. This method can be used in hand gesture recognition devices [24], in handicap assistant devices [25] and also in gaming devices [26].

1.3 Thesis Subject and Contribution

The subject of this thesis includes the design and construction of a prototype that contains an optical sensor and a three-axis accelerometer embedded in the same circuit board. The reason to have both sensors in the same board is to guarantee that they are under the same conditions and suffer the same displacement when moving the prototype in a 2D surface. In this way, it is possible to compare the output of both sensors by applying the same input (the user's interaction with the prototype).

In the computer side, the mouse driver software only requests the relative displacement in X and Y directions, which physically means the velocity in both coordinates. The extraction of the velocity from the optical mouse is straight-forward, since the sensor already returns the relative motion based on the successive images captured by its optical sensor. For inertial systems based on accelerometers, it is necessary to integrate the acceleration measured by the sensor. This integration can be performed in many ways. In this thesis, three digital signal processing methods are used to integrate the acceleration:

- Fuzzy-Neural Network Estimator;
- Kalman-Filter;
- State-Machine based Filter;

The performance of each technique is determined by comparing their resultant velocity curve of X and Y directions with the optical sensor resultant velocity curve. A multiple comparison is possible by collecting data from the accelerometers and optical sensor during some seconds and processing it off-line using a software application running in the computer.

After evaluating the performance of each integrator techniques, only the best ones are ported to run in the prototype, which has a limited microprocessor.

1.4 Outlines of Thesis

The content of this thesis is organized as follows.

Chapter 2: details about the design and the construction of the prototype used in this project are described. The description includes information about the main components used on the circuit board, as the microcontroller, optical and accelerometer sensors. It also includes the specification of the software running on the host and on the prototype.

Chapter 3: the integration techniques of the acceleration coming from the accelerometer sensor are described. For each technique, the mathematical model and details of the algorithm are presented.

Chapter 4: the experiment results are presented according to the developing steps of algorithms in chapter 3. Graphics containing the resultant velocity curve of each technique are presented, and the experimental results are discussed.

Chapter 5: the conclusion of this thesis and the possible improvement in the future is presented in this chapter.

Chapter 2. The Prototype

2.1 The Components

The prototype designed to test and evaluate an inertial mouse device based on accelerometers has three main components: the microcontroller, accelerometer sensor and optical sensor. All components are embedded in the same circuit board that was designed to connect to a host machine through a USB port. Each component will be explained in details on the next sections.

2.1.1 Microcontroller

The microcontroller PIC from Microchip with reference number 18F4550 is used in this prototype, which is responsible to read the sensors, manipulate the measured data and send the results to a host machine. The main reasons to use this microcontroller are because of the following features:

- Support to USB v2.0, a common protocol used in mouse devices when connecting to a computer. The USB port also supplies the power to all components embedded in the prototype;
- Analog Digital Converters (ADC), which are used to read the three-axis of the accelerometer sensor.
- Support to SPI protocol, which is used to control the optical sensor.

The oscillator crystal used to provide the clock signal to the microcontroller has 20 MHz of frequency. The ADCs from the microcontroller has resolution of 10 bits, allowing quantizing the analog signal from the accelerometers to a digital value that varies from 0 to 1024.

2.1.2 Three-Axis Accelerometer

The three-axis accelerometer used in this prototype is from Freescale Semiconductor and its reference number is MMA7360L. This sensor is a low power, low profile capacitive micro-machined accelerometer featuring signal conditioning, a 1-pole low pass filter, temperature compensation, self test and g-Select which allows the selection between 2 sensitivities[11].

On next, some of technical specification from this accelerometer is listed:

- 3mm x 5mm x 1.0mm LGA-14 Package
- Low current consumption: 400 μA
- Sleep Mode: 3 μA
- Low Voltage Operation: 2.2 V – 3.6 V
- High Sensitivity (800 mV/g at 1.5g)
- Selectable Sensitivity (±1.5g, ±6g)

For better performance in low accelerations, the most sensitive option is set (±1.5g),

2.1.3 Optical Sensor

The optical sensor used in this prototype is from Avago Technologies and its reference number is ADNB-6012-EV. This sensor is based on a laser diode, which allows operating on many surfaces that prove difficult for traditional LED-based optical navigation. It also has high-performance architecture, which is capable of sensing high-speed mouse motion – with resolution up to 2000 counts per inch.

The subcomponents of this optical sensor include:

- an optoelectronic sensor with CMOS technology;
- a lens base, which is used to attach the optoelectronic;

- laser diode (VCEL), which also is attach to the lens base;
- a clip to attach properly the laser diode to the lens base;
- a base plate, which is attached to the PCB of the whole prototype.

On Figure 1 all parts of the optical sensor are illustrated.

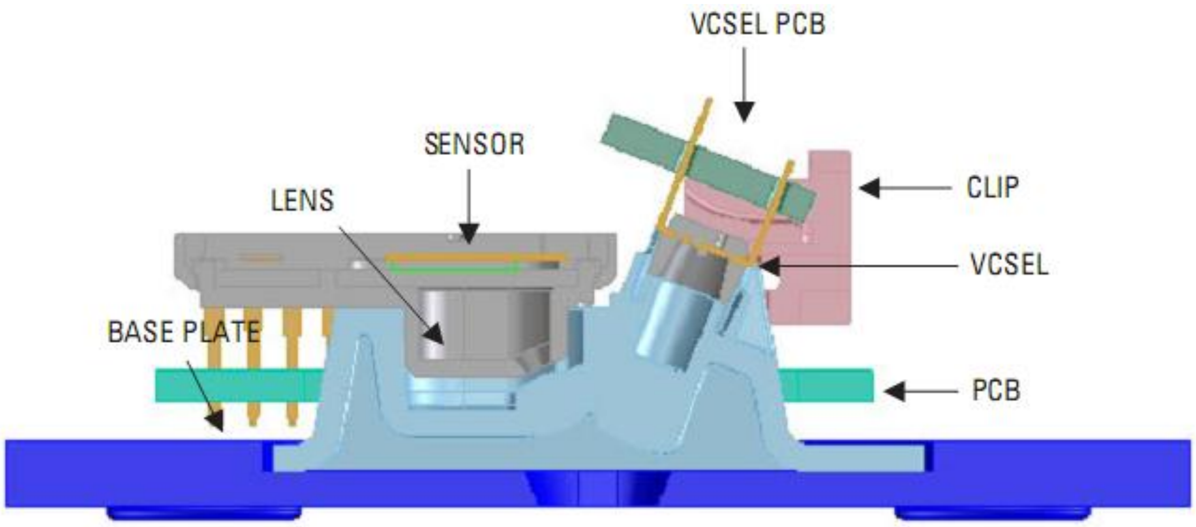


Figure 1: Cross section of a PCB assembly

2.2 The Layout and Physical Structure

The schematic and layout of the prototype were designed by using the software Protel 99. This program allows creating the schematic circuit including all components described in the previous sections and other elements, such as capacitors, resistors and voltage regulators. Once designed the schematic, the software also can generate automatically a PCB board including all circuit units and route the connection between them. The final PCB layout from the prototype is illustrated in Figure 2.

After the PCB is manufactured, all elements in the circuit are welded in the PCB board. Since the idea of this prototype is to behave as a mouse device, a physical structure of a commercial mouse was used to cover the PCB board. Also, a flat base plate was designed to be attached under the circuit board. In the base plate, there is an orifice, where the laser diode can reach the surface that it operates.

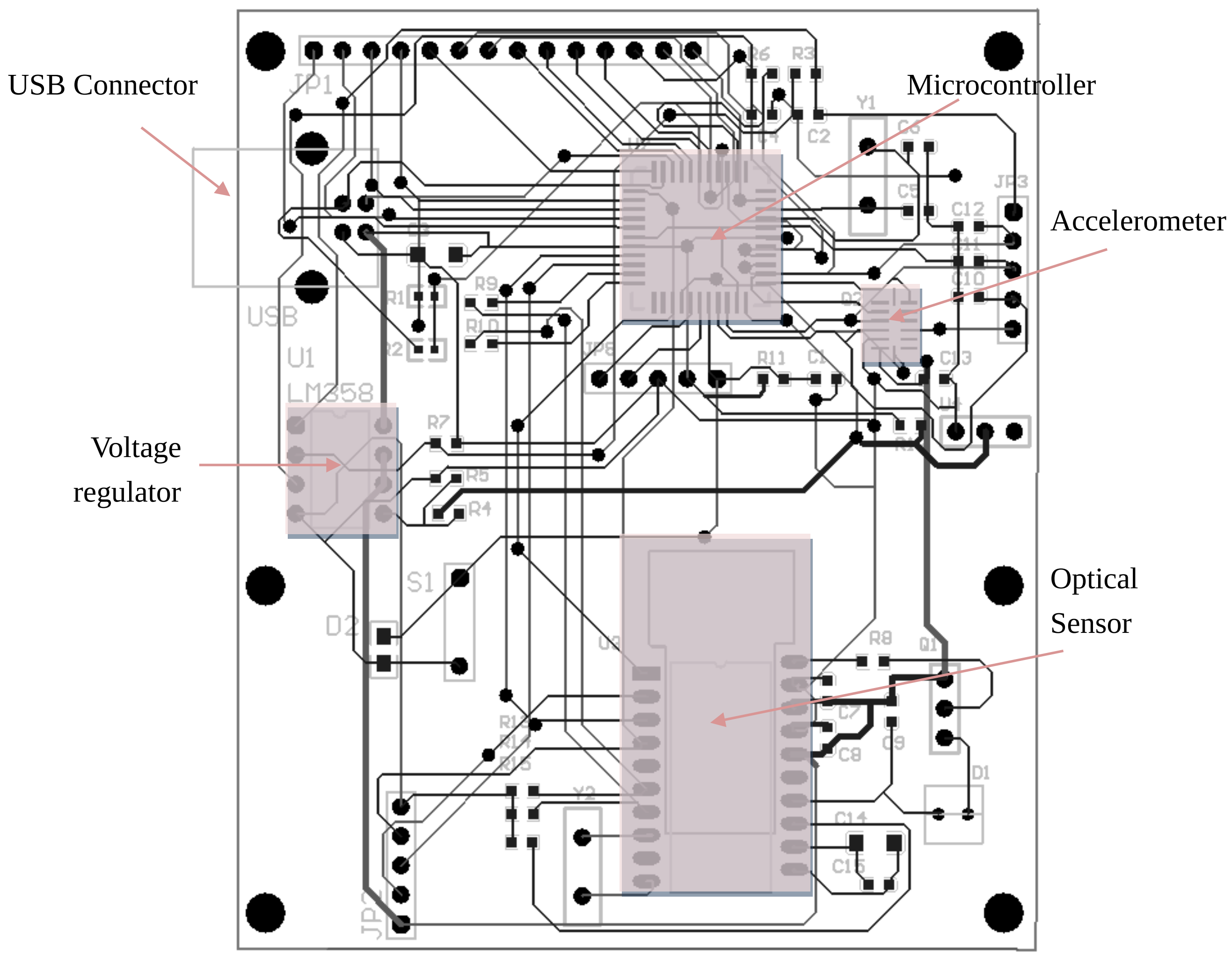


Figure 2: Prototype's layout

2.3 Software Development

2.3.1 USB Drivers and Device Classes

Any hardware device that interacts with a computer program must use a device driver, which is responsible to translate data between the operation system running in the computer and in the embedded system. In this project, the prototype represents the hardware device, which is connected to the computer using a USB port. In this way, the computer and the prototype must use a USB bus driver. Nowadays, many computer peripherals use an USB port to connect to a computer, such as printers, USB flash drives, webcams, keyboards and mouse devices. For each case, not only the USB Bus driver is used, but also a device class that specifies the device's functionality. In this project, two different device class based on USB are used:

- Communications device class (USB CDC), which provides an easy way to read and write any kind of data from/to an USB device. In this thesis, this device class is used to emulate a COM port, which will allow a software application running in the computer to manipulate an USB device as a RS-232 device. Therefore, a simple application can be implemented to read data from the prototype, specifically the sensors data processed by the microcontroller. A comparison between how the kernel and operating system treat a virtual COM port and a regular COM port is illustrated in Figure 3.

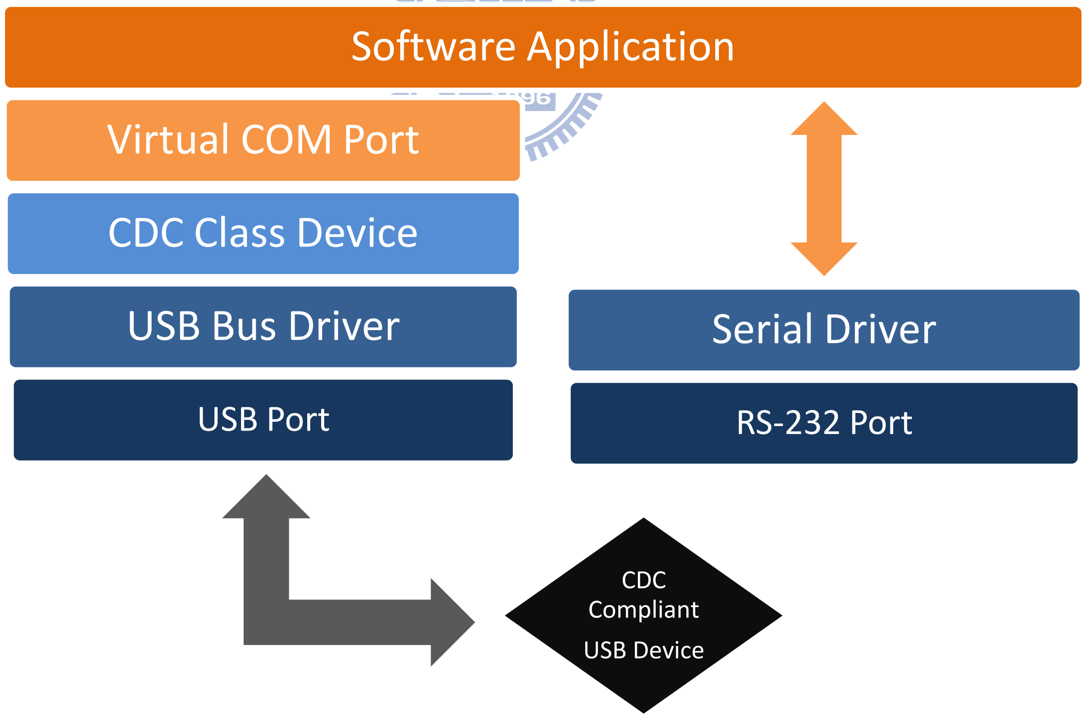


Figure 3: Hierarchy from Software Application to USB device

- Human Interface device class (USB HID), which stands for human interface devices, such as keyboards, game controllers and mouse devices. In this thesis, this device class is used to treat the prototype as a typical mouse device, which means the data sent from the prototype to the computer will be used to move the cursor on the screen. The packet format required when using the HID device class for a regular mouse device consists in 4 bytes:

- First byte: it is used to specify the state buttons of the mouse, (1 = pressed, 0 = not pressed), which allows processing simultaneously 8 different buttons. In the prototype, no buttons were added, since only the movement in X and Y directions are analyzed.
- Second byte: it is used to specify the displacement in the X coordinate. This is an 8 bit signed variable, which can assume values from -128 to 127. The value ZERO means that no displacement was detected.
- Third byte: it is used to specify the displacement in the Y coordinate. The same description from the X coordinate is applied here.
- Forth byte: it is used to specify the displacement from the mouse wheel. It is also an 8 bit signed variable, where the signal corresponds which direction the wheel was rolled.

2.3.2 Host Application

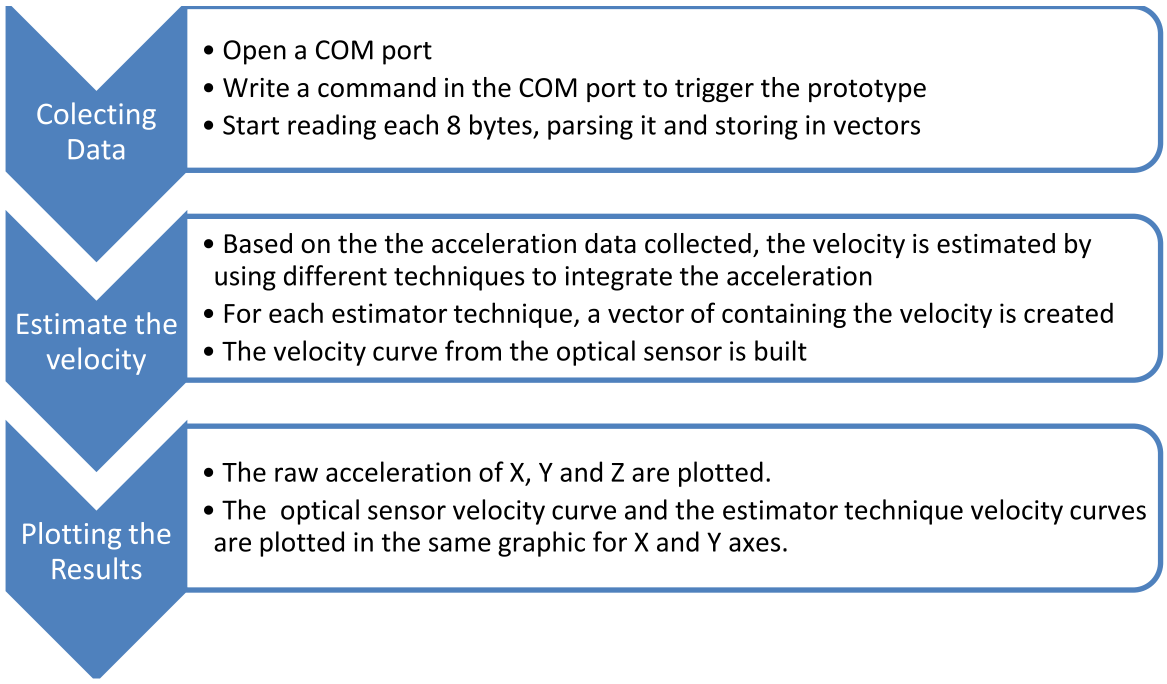
A host application was created to read the data from the prototype and process it for generating graphics based on different velocity estimators. This application is written in Matlab script, which can easily be used to plot large amount of data. For this case, the USB CDC device class is used. Therefore, the Matlab script application can connect to the prototype as a serial device, by using functions to open, read and write a serial port.

In this scenario, the prototype will be responsible to send binary packets with 8 bytes of size. The meaning of each byte is explained in Table 1.

1st byte	Most significant byte from X acceleration
2nd byte	Less significant byte from X acceleration
3rd byte	Most significant byte from Y acceleration
4th byte	Less significant byte from Y acceleration
5th byte	Most significant byte from Z acceleration
6th byte	Less significant byte from Z acceleration
7th byte	Reference Velocity X coordinate
8th byte	Reference Velocity Y coordinate

Table 1: Binary packet sent by the prototype

The last two bytes can refer to the optical sensor data or the velocity estimated by one of the integrator techniques implemented in the microcontroller. However, the integrator technique will be only implemented in the microcontroller after choosing the best option among the techniques implemented in Matlab, which has the following flow chart:



- Open a COM port
- Write a command in the COM port to trigger the prototype
- Start reading each 8 bytes, parsing it and storing in vectors

- Based on the the acceleration data collected, the velocity is estimated by using different techniques to integrate the acceleration
- For each estimator technique, a vector of containing the velocity is created
- The velocity curve from the optical sensor is built

- The raw acceleration of X, Y and Z are plotted.
- The optical sensor velocity curve and the estimator technique velocity curves are plotted in the same graphic for X and Y axes.

Based on the plotted velocity curves, it is possible to define which techniques present the best performance. Once the best techniques are defined, their algorithms will be implemented in the microcontroller embedded in the prototype. The same USB CDC driver and the same software application running in the computer can be used to validate the porting of this algorithm to the microcontroller. The embedded application only has to substitute the optical sensor measurements for the velocity estimated by the technique ported in the microcontroller. When plotting the graphic results using the host application, the curves from the ported estimate technique and the original implementation in Matlab must be similar. They will not be totally the same because the microcontroller has a limited architecture; implicating some parts of the algorithm are implemented with fix point representation. In Matlab, all variables are represented as floating numbers.

The script is structured in different files. Each file will be clarified on next:

- “runapp.m”, it is responsible to open the COM port and collect the data from the prototype;
- “plotall.m”, it will plot all graphics for data analysis;
- “kalman_filter.m”, this is a function file, which is responsible for estimating the velocity by using the Kalman filter. The output is a velocity vector and the input is the acceleration signal of one axis.
- “fuzzy_integrator.m”, this is a function file, which uses a fuzzy-neural network to estimate the velocity. The output is also a velocity vector and the input is the acceleration signal of one axis.
- “state_machine.m”, a function file that uses a series of different states to estimate the velocity. It has the same inputs and outputs from the other cases.
- “kalman_state.m”, it is a combination between Kalman filter and state machine solutions.

- "combined_sm.m", a function file that combine the state machines from axes X and Y, where the inputs are the acceleration signal of both axes and the output is the a velocity matrix containing the velocities in X and Y.

2.3.3 Embedded Application

The embedded application is written in C language and the compiler used is CCS C Compiler, which supports Microchip PIC 18x series. One of the advantages of using this compiler is its support to different USB device classes, including HID USB and CDC USB device classes. Therefore, a simple API is provided to access the USB driver. The applications running in the prototype can be developed and compiled in the host machine, and the resultant firmware is downloaded to the target by using a programmer provided by Microchip called MPLAB ICD 2, which uses JTAG protocol to transfer the binary file from the host machine to the microcontroller PIC.

The main program to be executed in the microcontroller consists in a main loop flow that reads the sensor data, process it and send the result to the host machine. The sensor readings are split in two parts. The first part reads the optical sensor using the SPI protocol. The resultant data is two bytes; each byte represents one of the XY coordinates. The second part is responsible to read the accelerometer sensor by using three analog-to-digital converters from the microcontroller. For each axis (XYZ), eight consecutive measures are made and the mean value of them is used to smooth the results. The microcontroller only allows selecting one ADC channel at a time, and for each measurement a delay must be specified to quantize the correct value of the acceleration, which will define the sampling rate of the measurements. This delay is defined in agreement with the maximum sampling rate allowed in the ADC from the microcontroller. Besides that, the delay cannot be too short, or the accuracy of the result will be distorted. Also, it cannot be too long, or the final number of packets per second send to the host machine will be too small. For instance, the number of packets per second that a commercial mouse device sends to the computer is around 100 packets per second, where each packet corresponds to the format commented in section 2.3.1. Therefore, considering the time to measure all axes is 15% of the time between two consecutive packets, the sampling time of the accelerometers should be smaller than 1.5 ms, which will let around 8.5 ms for the embedded application process the data and integrate the acceleration. In this way, the quantize time for each ADC measurement is defined to 50 μs.

The sampling time line is illustrated in the Figure 4.

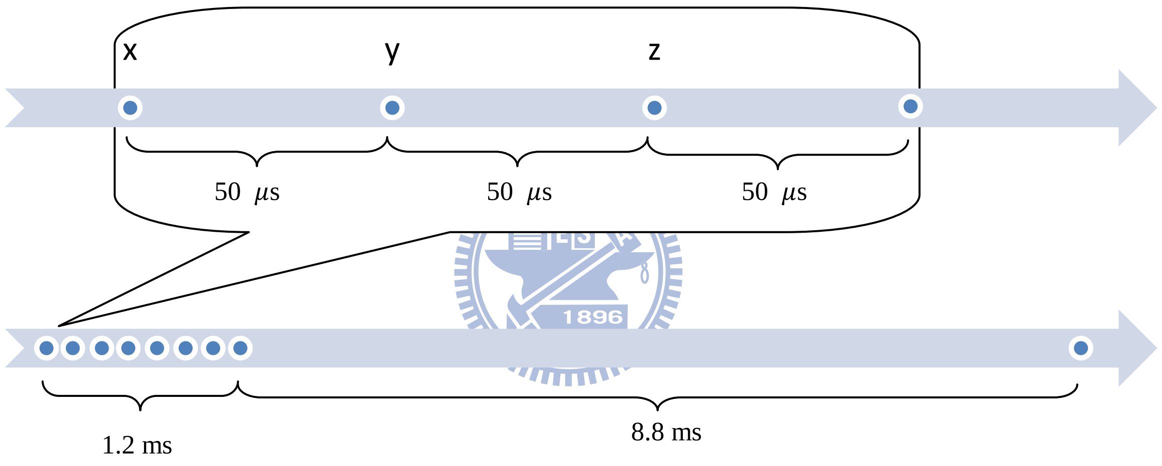


Figure 4: Sampling time line

Note that the time to process the data and integrate the acceleration is not necessary 8.8 ms; that graphic is only an estimation of the minimum sample rate necessary to similarly perform as the commercial mouse devices.

The flow chart of the main program will be explained in details on next. Some parts of the flow will be explained in the next chapter.

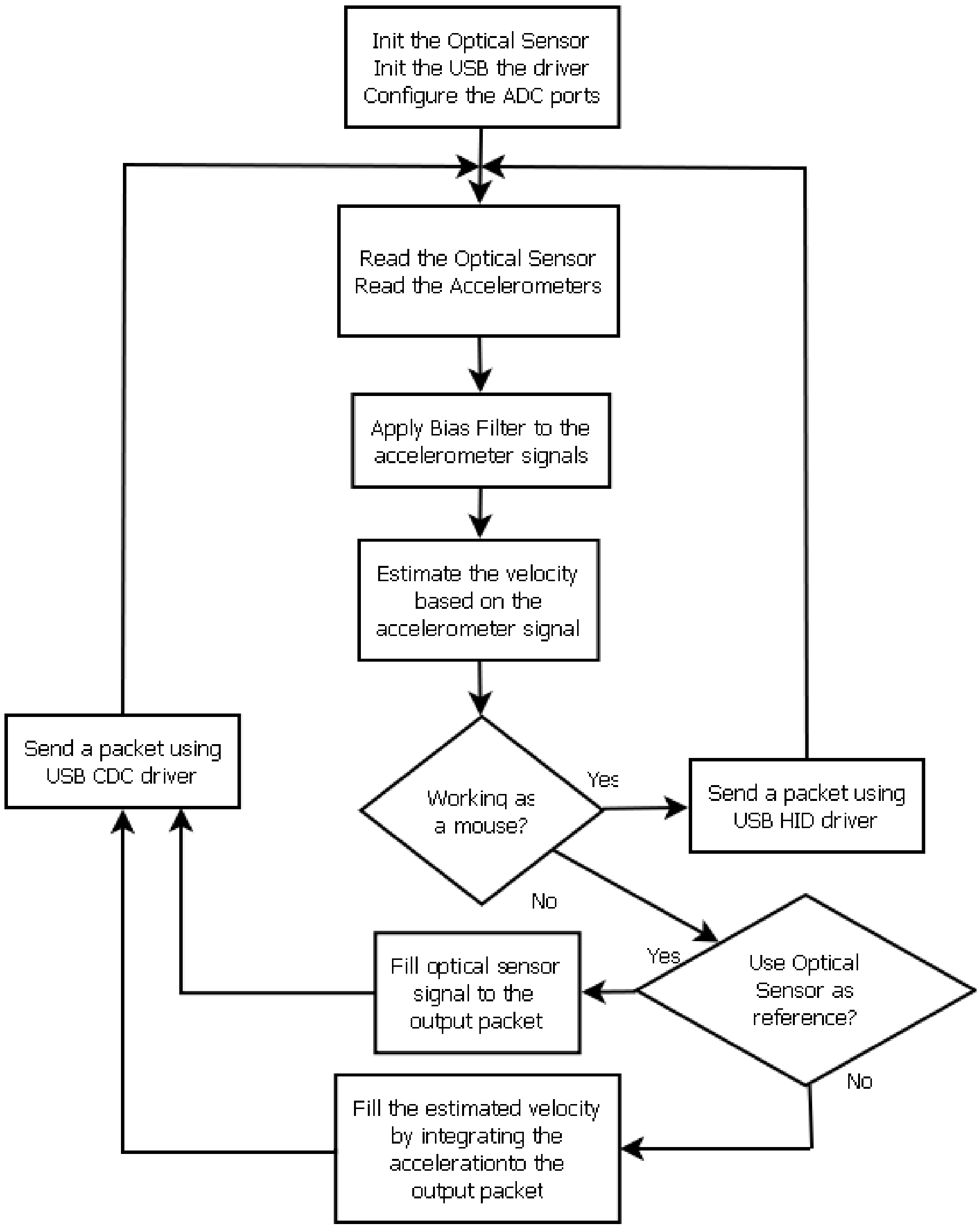

Init the Optical Sensor
Init the USB the driver
Configure the ADC ports
Read the Optical Sensor
Read the Accelerometers
Apply Bias Filter to the accelerometer signals
Estimate the velocity based on the accelerometer signal
Working as a mouse?
Yes
Send a packet using USB HID driver
No
Use Optical Sensor as reference?
Yes
Fill optical sensor signal to the output packet
No
Fill the estimated velocity by integrating the accelerationto the output packet
Send a packet using USB CDC driver

Figure 5: Flow chart from the embedded application

Chapter 3. Signal Processing Methods

3.1 Data Preparation

Before using any technique to integrate the acceleration signal coming from the three-axis accelerometer, it is necessary to prepare the data by applying some statistical techniques, filters and calibration.

3.1.1 Statistical Techniques

The first step, already commented in the section 2.3.3, is to obtain the average of N acceleration values collected in a high speed sample rate. The following equations are used to obtain a more accurate acceleration:

$$\bar{a}_x = \frac{1}{N}\sum_{i=1}^{N} a_x^i \quad , \qquad where\ a_x^i is\ the\ i^{th}\ raw\ acceleration\ from\ X\ axis$$

$$\bar{a}_y = \frac{1}{N}\sum_{i=1}^{N} a_y^i \quad , \qquad where\ a_y^i is\ the\ i^{th}\ raw\ acceleration\ from\ Y\ axis$$

$$\bar{a}_z = \frac{1}{N}\sum_{i=1}^{N} a_z^i \quad , \qquad where\ a_z^i\ is\ the\ i^{th}\ raw\ acceleration\ from\ Z\ axis$$

In this project, the number of data N collected each time is equal 8. Another formula applied in the signal collected in high speed sample rate is the amplitude filter, which returns the difference between the maximum and minimum values. This filter is applied exclusively for the Z axis:

$$Amp_z = \max_{1 \le i \le N} [\, a_z^i] - \min_{1 \le i \le N} [\, a_z^i]$$

$$where\ a_z^i\ is\ the\ i^{th}\ raw\ acceleration\ from\ Z\ axis$$

The value of Amp_z can be used as an indicator of the prototype' state, since the Z axis can measure the vibration of the environment. Usually this vibration is a high frequency noisy signal. However, when the prototype is moving, this vibration tends to increase, which means the amplitude will be higher. This effect can be observed in the Figure 6. The highlighted areas indicate when the prototype is moving.

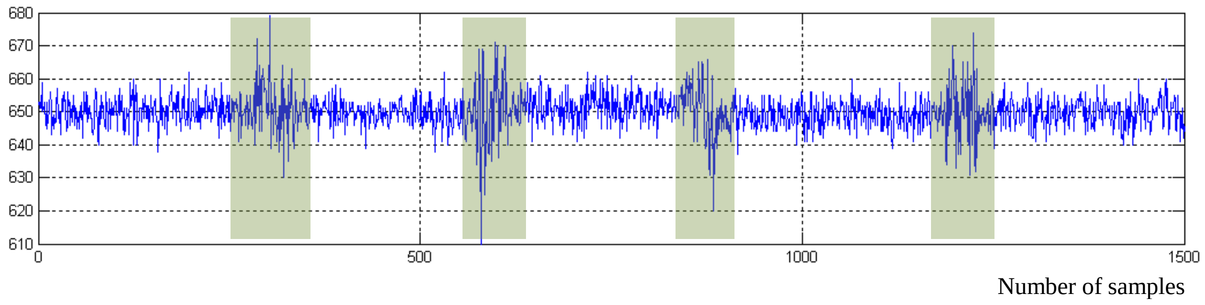


Figure 6: Raw acceleration from Z Axis (no scale)

3.1.2 Bias filtering

All axes from the accelerometer have an output signal bias which can be observed when no acceleration is applied in that axis. However, if the acceleration of gravity is considered, it can influence in the bias factor of all axis. Since the resolution of the ADC is 10 bits, the acceleration can assume values from 0 to 1024. Ideally, if the accelerometer is not under influence from any force, the acceleration should be around the median value of that scale (512 without scale). However, since the Z axis is in the same direction of the gravity force, its bias will be higher than the other axes, as you see in Figure 7. The average acceleration in X axis of that group of data is 501.34; in Y axis is 537.99; and in Z axis is 650.12.

An inertial mouse device working in a flat surface only requires the integration of X and Y accelerations. Therefore, it is necessary to apply some kind of biased filter to remove the bias factor in both X and Y acceleration signals. A simple way to compensate the signal is to subtract the raw acceleration by the average value measured when the prototype is stationary. However, this solution may not work well in case the bias factor changes dynamically, which is a common behavior when moving the device in a not totally flat surface.

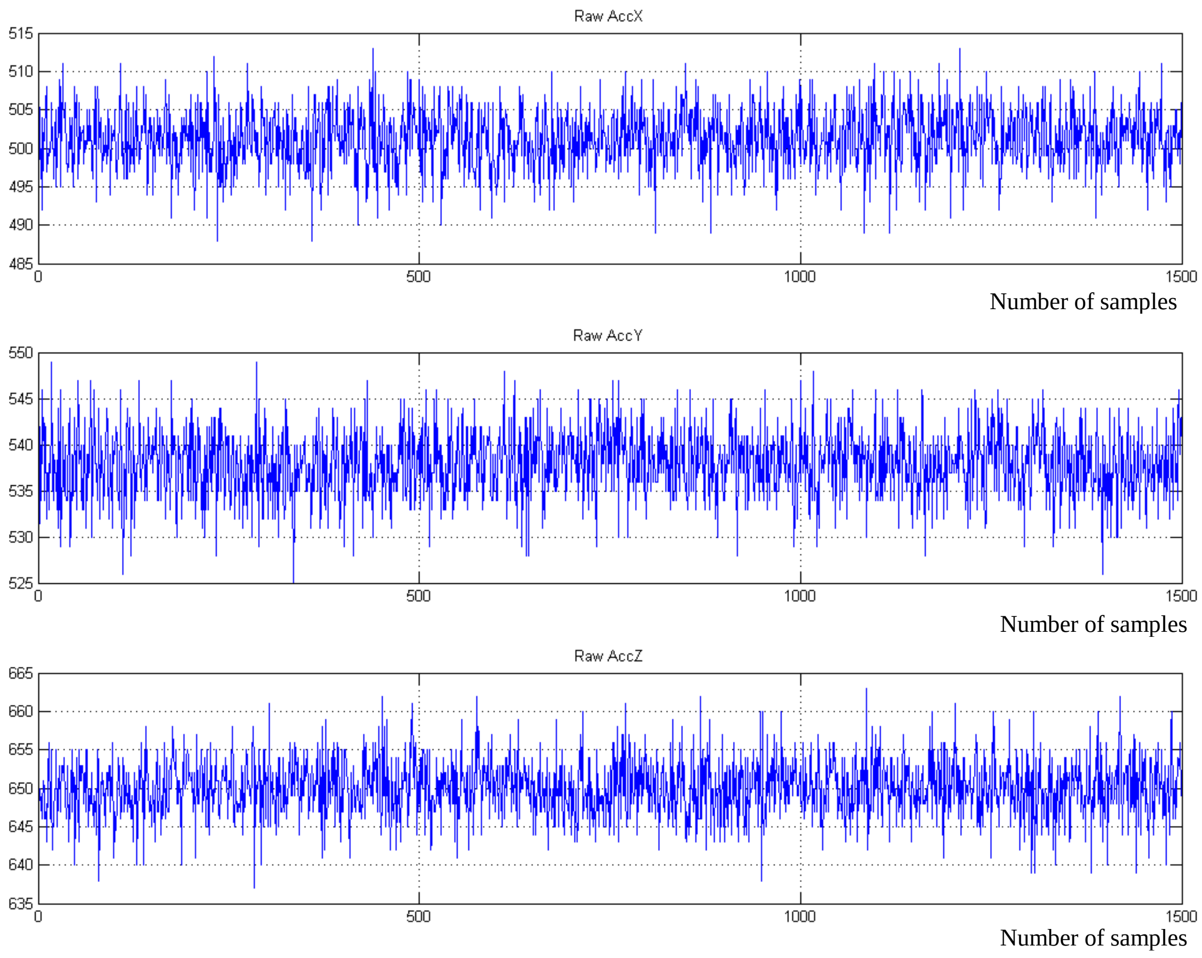


Figure 7: Raw acceleration of XYZ axes without moving the prototype

An experimental example of the problem when applying a constant value to compensate the bias factor is illustrated in Figure 8. In that scenario, moving the prototype in one direction just few centimeters was enough to dynamically change the bias factor. The explanation of this variance is related to the tilt angle, which is the angle between the Z axis of the accelerometer and the force of gravity. If the tilt angle is equal zero, the gravity force will not influence the X and Y bias factor. However, if the tilt angle is different than zero, the gravity force will influence the X and Y bias factor proportionally to the tilt angle.

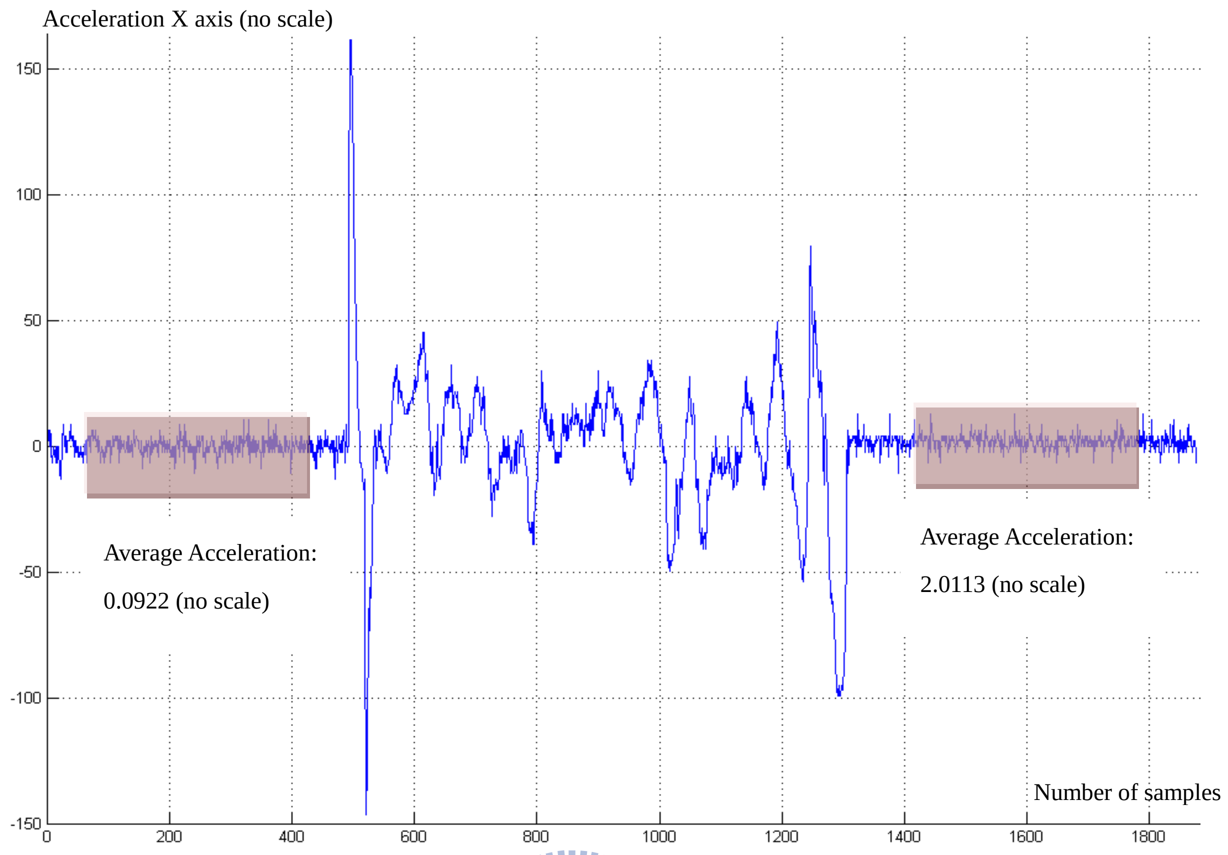


Figure 8: After constant bias compensation

The proposed solution to contour this problem consists in using an Impulse Infinite Response Filter (IIR Filter), which will eliminate the low frequencies and make the acceleration signal to converge slowly to zero. The transfer function in Z domain can be represented as:

$$H(z) = \frac{1 - z^{-1}}{1 - \delta z^{-1}}$$

Considering the raw acceleration $\bar{a}$ as the input, the final output after applying the filter $H(z)$ is the unbiased acceleration $\hat{a}$. In the discrete time domain, the relation between the input and output can be represented as:

$$\hat{a}[n] = \delta \cdot \hat{a}[n-1] + (\bar{a}[n] - \bar{a}[n-1])$$

In this formula, the coefficient δ determines how fast the unbiased acceleration converges to zero. If the coefficient δ is smaller and close to 1, the influence of the filter is reduced. The behavior of different δ in the filter can be observed in the Figure 9 . The same data from Figure 8 is used in this analysis.

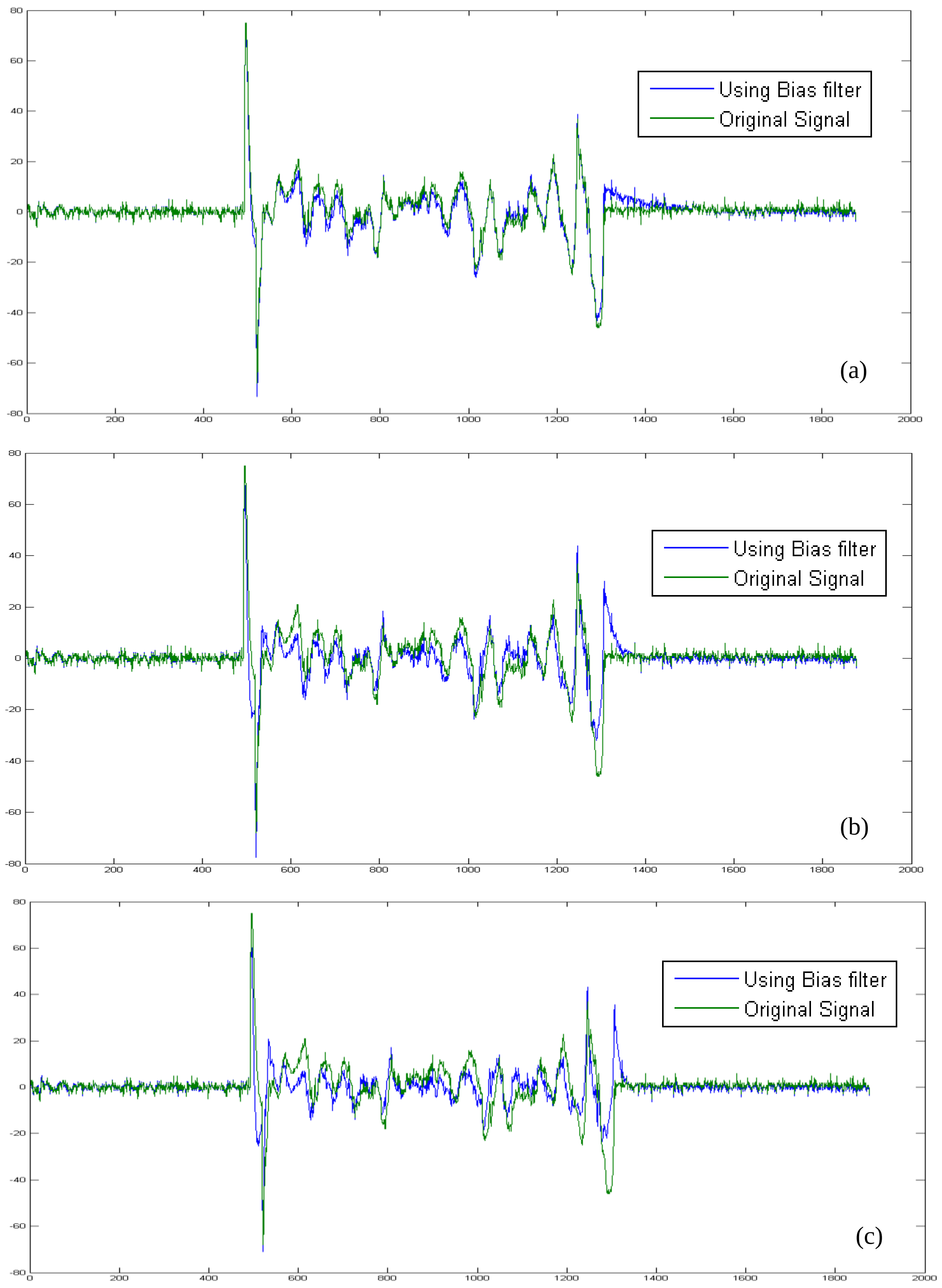


Figure 9: Unbiased acceleration after applying filter H(z) with different δ:

a) $\delta = 0.99$, b) $\delta = 0.95$, c) $\delta = 0.90$

Comparing the different scenarios when changing δ, it is noticed that the case (a) from Figure 9, the distortion is smaller than the other scenarios during the movement. However, in the end of the movement, a small overshoot is observed and the signal converges slowly to zero when comparing with smaller δ. A closer look of the acceleration right before the end of the movement can be observed in the Figure 10.

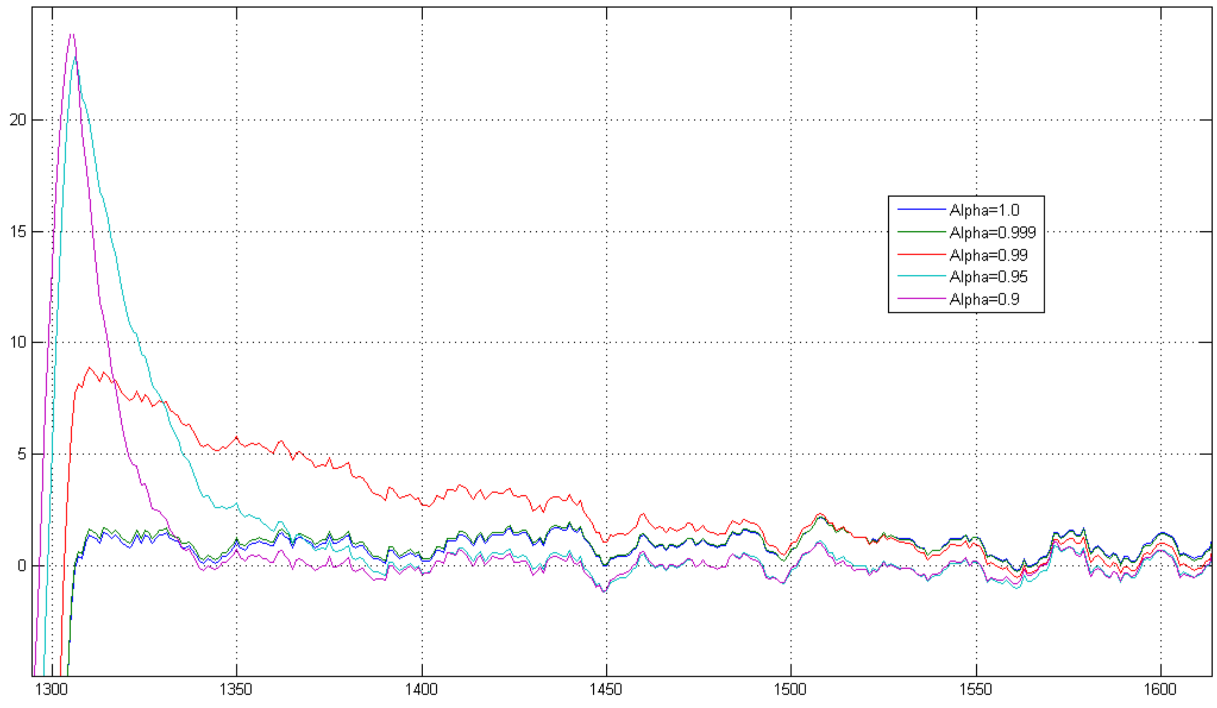


Figure 10: Closer look in the end of the movement

An optimal solution would be to have an adaptive δ, that increases when some movement is detected and decreases when there is no movement. In this way, the signal will not be distorted too much during the movement, and in case there is any overshoot in the end of the movement, the filter will converge the acceleration to zero really fast, since δ will decrease after the displacement. Any movement from the prototype can be easily detected when calculating the average deviation of the signal (next section presents the mathematical formula). If the average deviation is large, the most probably state of the prototype is in movement, otherwise, would be stationary. An example using an adaptive δ in the biased filter is show in the Figure 11 . Again, the same data from the previous experiments is used. In this scenario, the δ changes between 0.9 and 1.0, depending on current average deviation of the acceleration signal. When calculating the average deviation, only the last 15 points were considered.

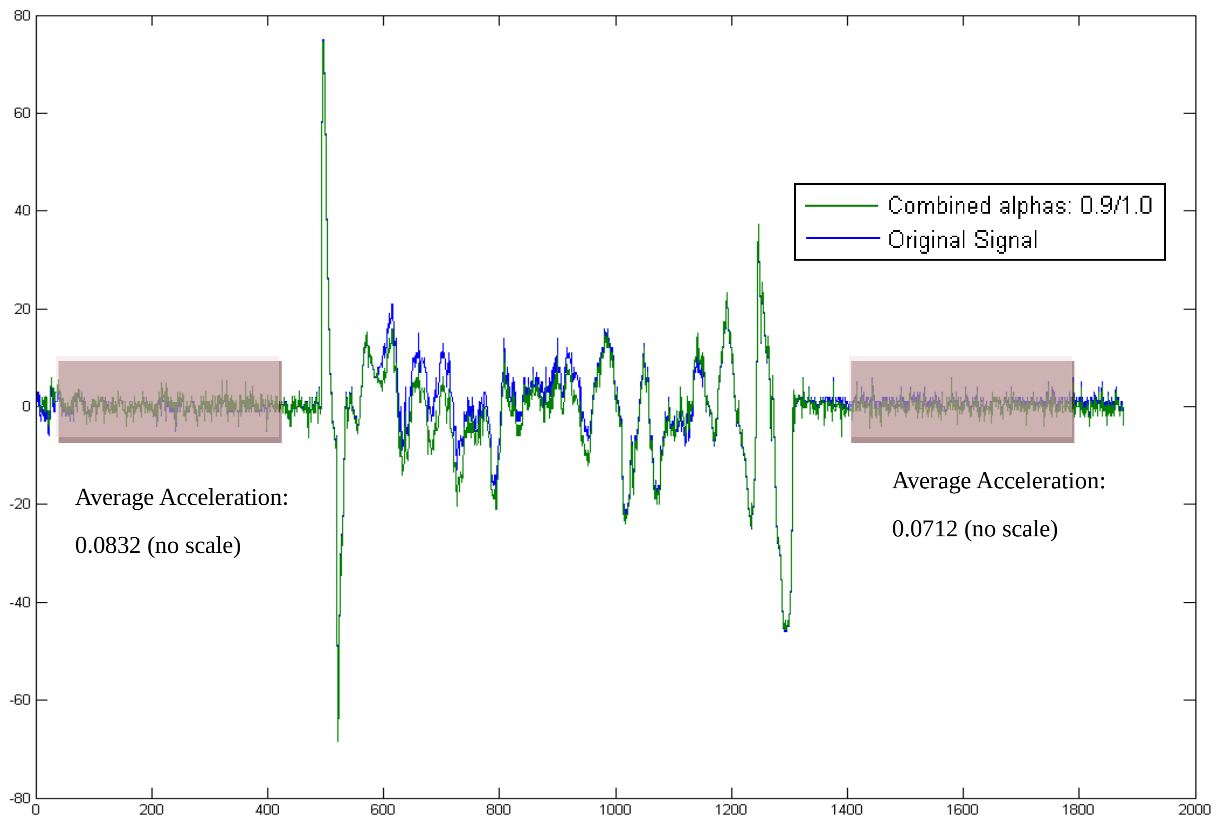


Figure 11: Adaptive biased filter

3.1.3 Calibration Process

Another important step before apply any estimator technique is the calibration process. In this procedure, some relevant parameters are extracted from the system, which will be used as reference to better estimate the velocity.

Average Absolute Deviation for each axis

The initial task to perform is to identify the average absolute deviation of the all axis from the accelerometer when the prototype is stationary. The formula of the average deviation can be expressed as:

$$AD(m) = \frac{1}{N}\sum_{i=1}^{N} |a_i - m|, \qquad where\ m = \frac{1}{N}\sum_{i=1}^{N} a_i$$

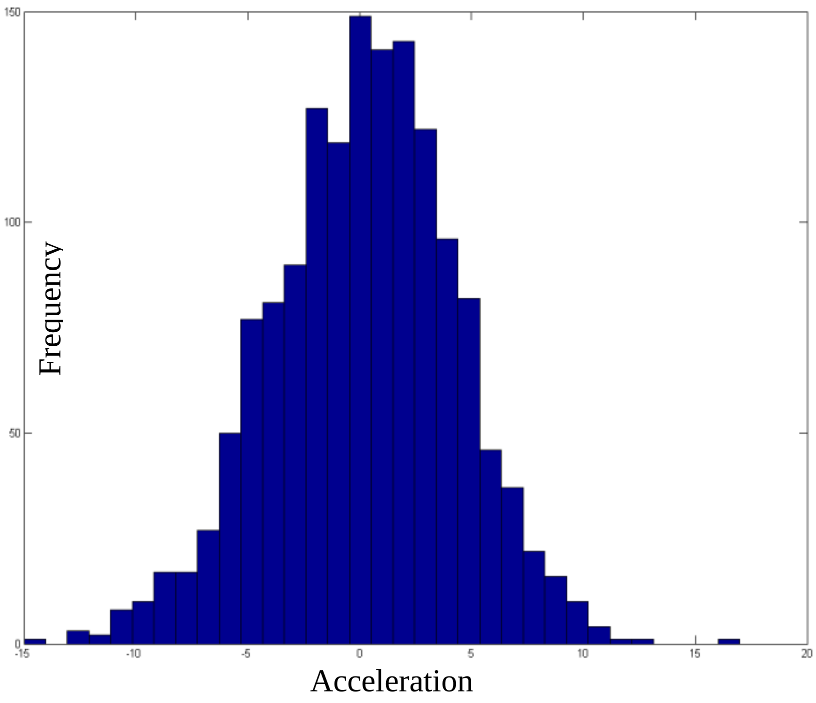
This formula should be applied with $N = 15$ for a huge amount of data for each axis (at least 500 samples). The overlap between the points should be considered. Therefore, the formula can be represented in the discrete domain as:

$$AD[n] = \frac{1}{N}\sum_{i=0}^{N-1}\left|a[n-i] - \frac{1}{N}\sum_{k=0}^{N-1} a[n-k]\right|$$

Calculating the histogram from the resultant data, it is possible to extract the most common value of the average absolute deviation, which will be used as one of the calibration parameters.

Discrimination Windows for each axis

During the calibration process, the biased filter can be used to converge the acceleration signal rapidly to zero by setting a small δ. Once the acceleration reach some value near to zero, δ is switched to value close to one. In this scenario, the acceleration will assume values around zero when the mouse is stationary, and the noise of the specific axis can be measured. Calculating the histogram of the acceleration after applying the biased filter (note that the average acceleration will be around ZERO), it is possible to identify the amplitude of the noise.



From the histogram, the acceleration value that corresponds to 10% of the most common acceleration level is obtained.

For example, in the histogram on left, the most common acceleration value is around 0 unit, counting 150 times. 10% of 150 is 15, which corresponds to the acceleration of approximately 10 units.

This acceleration (10 units) will define the discrimination window of the noise. That means any acceleration between -10 and 10 will discriminate as a noise.

Average Value for Z axis

Since the biased filter is only applied for X and Y, only the average value of Z is calculated. Its calculation is straight-forward, the mean value of at least 500 samples from Z axis is obtained when the prototype is stationary.

3.2 The Fuzzy-Neural Integrator

The first technique used to estimate the velocity from the accelerometer signals is based on the fuzzy-neural network. The reason to use this specific combination of two fields – fuzzy systems and neural networks – is because the synergistic integration of them will bring many benefits from both fields. The neural networks provide connectionist structure and learning abilities to the fuzzy logic systems, and the fuzzy logic systems provide the neural networks with a structural framework with high-level fuzzy IF-THEN rule thinking and reasoning. In the theory, there are many possible ways to integrate fuzzy systems and neural networks. The one used in this thesis is called "Fuzzy Modeling Networks", which the basic idea is to realize the process of fuzzy reasoning by the structure of a neural network and express the parameters of fuzzy reasoning by the connection weights of a neural network. Therefore, it can automatically identify the fuzzy rules and tune the membership functions by modifying the connection weights of the networks using the back-propagation learning algorithm**Error! Reference source not found.**.

The configuration type of the Fuzzy Modeling Network is shown in Figure 12. In this particular network, only three inputs are used. Later, a different version of the

Fuzzy-Neural Network will be introduced, including a different number of inputs and nodes; however all of them share the same configuration that will be explained now.

The Fuzzy Modeling Network can be divided into premise part and consequent part. The premise part consists in two layers. The first layer corresponds to the number of inputs of the system, where each input represents a node. The second layer corresponds to the membership functions of each input, where each node presents a fuzzy variable from the membership functions.

The consequence part also has two layers. The third layer corresponds to the fuzzy rules, where each node is the relation between different membership functions. The fourth layer is the output layer, where a unique node represents the final output. The consequence part can be represented as a fuzzy singleton:

$$R^i: IF\ x_1\ is\ A_1^i\ AND\ x_2\ is\ A_2^i\ AND\ x_3\ is\ A_3^i, \qquad THEN\ y = w_i$$

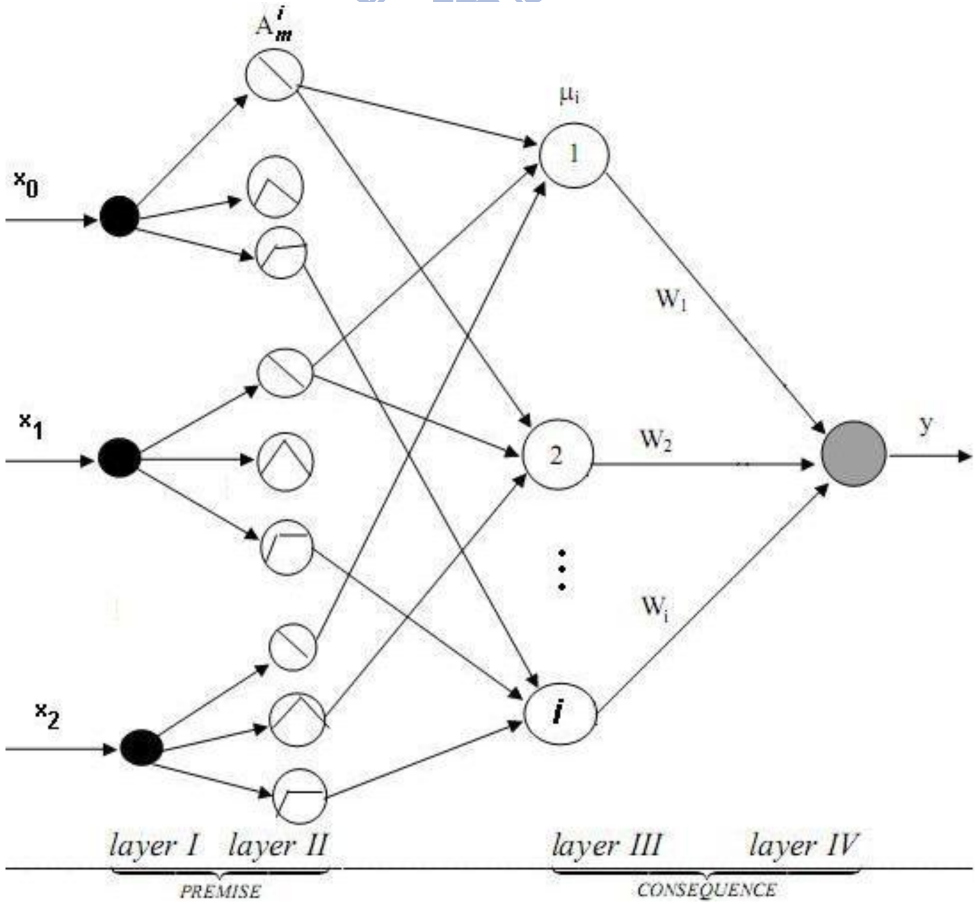


Figure 12: Configuration of the Fuzzy Neural Network

On the following sections, the modeling of the Fuzzy Neural Network is explained, including the mathematical model of parts of the network and the learning strategy.

3.2.1 Membership functions

A membership function is defined as the probability of any value from a physical or statistical quantity, such as acceleration, velocity or average deviation, belongs to a specific fuzzy set. Each membership function can contain one or more fuzzy sets, which are related to a linguistic variable. A fuzzy set is a pair (A, m) where A is a set and $m: A \rightarrow [0,1]$. On next, some examples of linguistic variable are shown:

- Acceleration：{POSITIVE, ZERO, NEGATIVE}
- Velocity：{POSITIVE, NEGATIVE}

Based on those linguistic variables, it is possible to define the fuzzy sets that will belong to the membership functions of each physical/statistical quantity.

Acceleration

The fuzzy sets that represent each linguistic variable from the acceleration can be expressed mathematically as:

Consider the following parameters are based on the extraction of the discrimination window (A_{dw}) *during the calibration process explained in section 3.1.3 . The discrimination window represents the interval as* $-\boldsymbol{A_{dw}} < a < \boldsymbol{A_{dw}}$ *, were* $\boldsymbol{a}$ *is the acceleration value.*

$a_{min} = \frac{\boldsymbol{A_{dw}}}{\mathbf{2}}$	$a_{max} =$ *300~400 units maximum acceleration*	$a_{zro} = \frac{\mathbf{3}}{\mathbf{4}}\boldsymbol{A_{dw}}$

Positive Acceleration	$A_{pos}(a) = \begin{cases} 0 & , \quad a \le a_{min} \\ \dfrac{a}{a_{max} - a_{min}} + \dfrac{a_{min}}{a_{min} - a_{max}}, & a_{min} < a < a_{max} \\ 1 & , \quad a \ge a_{max} \end{cases}$
Zero Acceleration	$A_{zro}(a) = \begin{cases} \dfrac{a}{a_{zro}} + 1 & , \quad -a_{zro} < a \le 0 \\ \dfrac{-a}{a_{zro}} + 1 & , \quad 0 < a < a_{zro} \\ 0 & , \quad otherwise \end{cases}$
Negative Acceleration	$A_{neg}(a) = \begin{cases} 1 & , \quad a \le -a_{max} \\ \dfrac{-a}{a_{max} - a_{min}} + \dfrac{a_{min}}{a_{min} - a_{max}}, & -a_{max} < a < -a_{min} \\ 0 & , \quad a \ge -a_{min} \end{cases}$

Graphically, the above equations represent the following membership function:

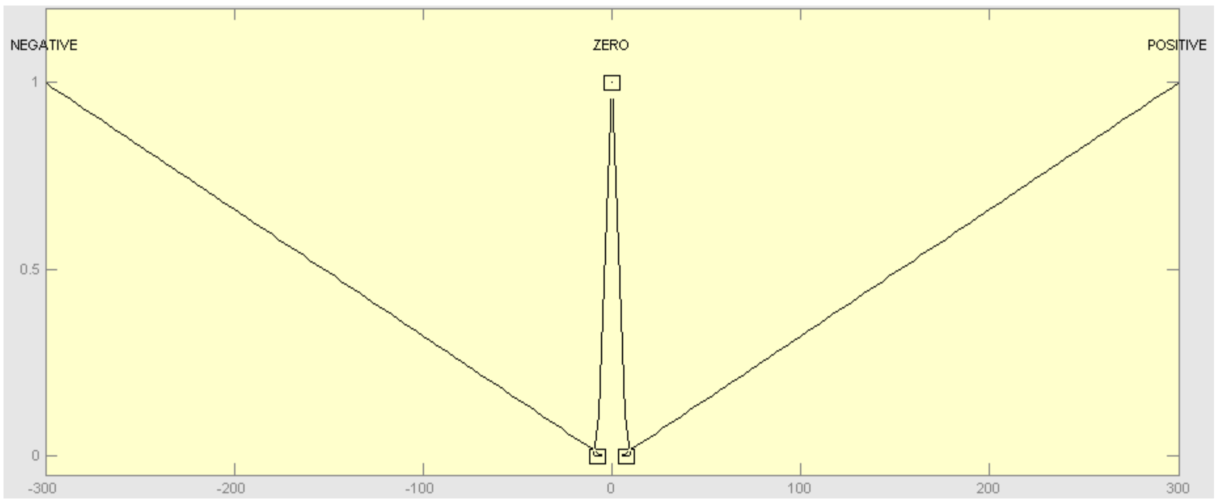


Figure 13: Acceleration Membership Function

Velocity

The fuzzy sets that represent each linguistic variable from the velocity can be expressed mathematically as: (where **v** is the velocity value)

Positive Velocity	$V_{pos}(v) = \begin{cases} 0 \quad , & v \le -1 \\ \frac{v}{2} + \frac{1}{2} \, , & -1 < v < 1 \\ 1 \quad , & v \ge 1 \end{cases}$
Negative Velocity	$V_{pos}(v) = \begin{cases} 1 \quad , & v \le -1 \\ \frac{-v}{2} + \frac{1}{2} \, , & -1 < v < 1 \\ 0 \quad , & v \ge 1 \end{cases}$

Graphically, the above equations represent the following membership function:

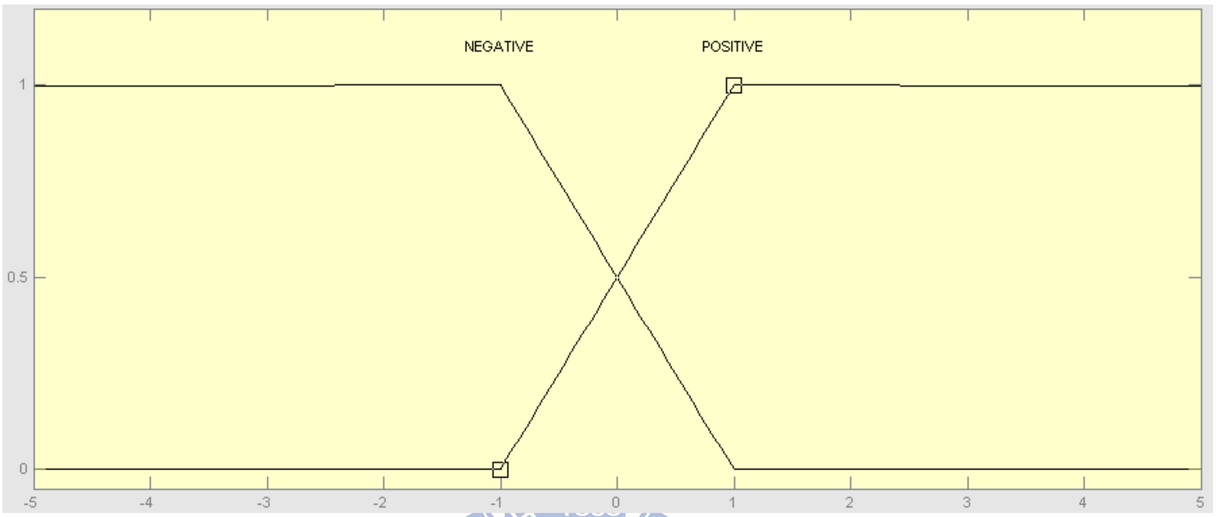


Figure 14: Velocity Membership function

3.2.2 Fuzzy Rules

The fuzzy rules define the relation between the different membership functions. As already commented, singleton rules are used to elaborate the relationship between them:

$$R^i: IF\ x_1\ is\ A_1^i\ AND\ x_2\ is\ A_2^i\ AND\ x_3\ is\ A_3^i, \qquad THEN\ y = w_i$$

Where $x_1, x_2\ and\ x_3$ are the inputs of the fuzzy-neural network. And A_m^i is the result of a specific fuzzy set of the membership function of each input m, with $m = 1,2,3$, in case the fuzzy-neural network has only three inputs. Mathematically, the result of the node can be expressed as:

$$\mu_i = A_1^i \cdot A_2^i \cdot A_3^i$$

The values of w_i is defined in agreement with the singleton rules. For example, imagine a Fuzzy-Neural Network with 3 inputs – the last three accelerations, defined as a_1, a_2, a_3. And the output is the difference velocity. Using the membership function for acceleration explained in the previous section, the following sentences can be written:

IF a_1 is *Positive* AND a_2 is *Positive* AND a_3 is *Positive*, THEN $w_0 = +3$

IF a_1 is *Negative* AND a_2 is *Negative* AND a_3 is *Negative*, THEN $w_1 = -3$

IF a_1 is *Negative* AND a_2 is *ZERO* AND a_3 is *ZERO* , THEN $w_2 = -1$

Note that the attribution of the weights w_i is based on physical behavior of a particle submitted to some acceleration.

3.2.3 Defuzzification

Defuzzification is a mapping from a space of fuzzy control actions defined over an output universe of discourse into a space of non-fuzzy control actions **Error! Reference source not found.**. The output signal based on the configuration from Figure 12 can be calculate as:

$$y^* = \sum_{i=1}^{N} \mu_i w_i \Big/ \sum_{i=1}^{N} \mu_i$$

Where μ_i is the node values of each fuzzy rule; w_i is weight of each connection bewteen the fuzzy rule nodes and the output.

3.2.4 Back Propagation Algorithm

The back-propagation learning algorithm is one of the most useful learning techniques used in neural networks. This learning algorithm is applied to multilayer feed forward networks consisting of processing elements with continuous differentiable activation functions.

To better understand how this technique can be used in this project, the following inputs and outputs are defined:

Inputs		Output	
Current Acceleration:	a[n]	Velocity Difference:	$dv^*[n]$
Previous Acceleration:	a[n-1]		
Previous Acceleration:	a[n-2]		
Current Velocity:	v[n]		

Where the accelerations are the measurements coming from the accelerometers and the current velocity is the integration of the acceleration using the Fuzzy-Neural Network.

When using back-propagation algorithm, it is necessary to have the desired output of the system, which will be represented by the optical sensor output. There are 3 phases when using this technique – collecting data, training the network and validation. Each phase will be explained on next.

Collecting Data

In this phase, the training data is created by collecting the last three acceleration signals from accelerometers, the current velocity and the difference velocity from the optical sensor when moving the prototype in one axis. This procedure is performed for X and Y axis. The block diagram of this phase is illustrated in Figure 15.

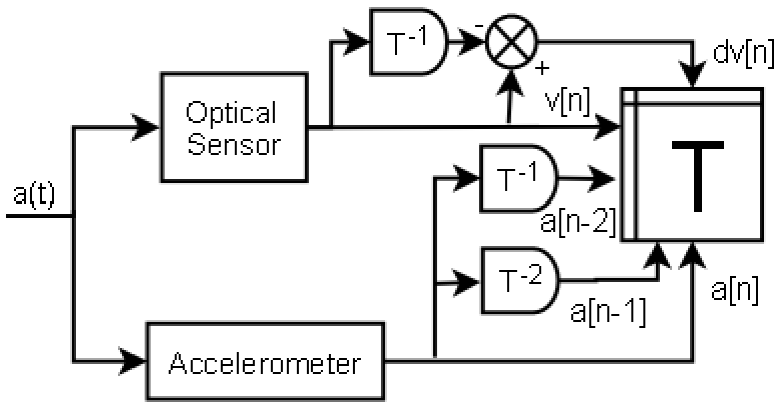


Figure 15: Block Diagram from Collecting Phase

Training the Network

Once the training data is collected, the Fuzzy-Neural Network is fed with this data. The network's output is compared to the desired output from the same training data. The error is calculated and propagated back to the network and the weights of each neuron are adjusted. This procedure is illustrated in the Figure 16.

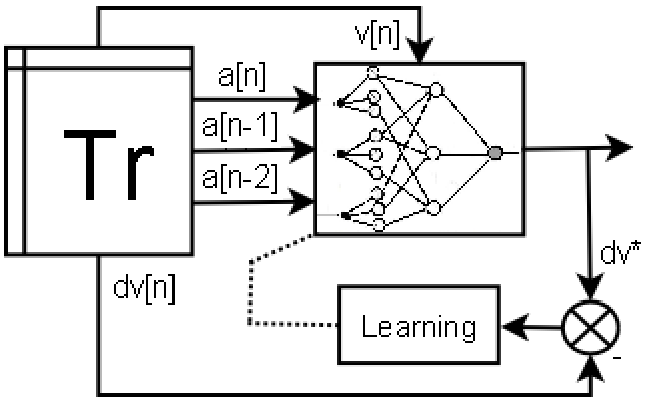


Figure 16: Block Diagram from Training Phase

The equations of the learning algorithm applied in this Fuzzy-Neural Network are explained in the following lines:

The output error measure (ε) is calculated as:

$$E = \frac{1}{2}(dv^*[n] - dv[n])^2$$

$$\varepsilon = \frac{\partial E}{\partial dv^*} = dv^* \text{-}\ dv[n]$$

The error is propagated backward to update the weights based on the learning rate η:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Splitting in partial differential equations:

$$\Delta w_i = -\eta \frac{\partial E}{\partial dv^*} \frac{\partial dv^*}{\partial w_i}$$

The output dv^* can be substituted by the equation described in section 3.2.3 and $\frac{\partial E}{\partial dv^*}$ represents the output error measure ε:

$$\Delta w_i = -\eta\varepsilon \frac{\partial\left(\frac{\sum_{i=1}^{N}\mu_i w_i}{\sum_{i=1}^{N}\mu_i}\right)}{\partial w_i}$$

$$\Delta w_i = -\eta\varepsilon$$

This is the final equation to adjust the weights of the network part of the Fuzzy-Neural Network.

Validation

In this phase, the output from the trained Fuzzy-Neural Network is compared with the optical sensor signal. If $v^*[n] - v[n] < E_{max}$, where E_{max} is the maximum error acceptable, and $v^*[n] = dv * [n] + \ v^*[n-1]$. If the average error is below the threshold, no more training is necessary.

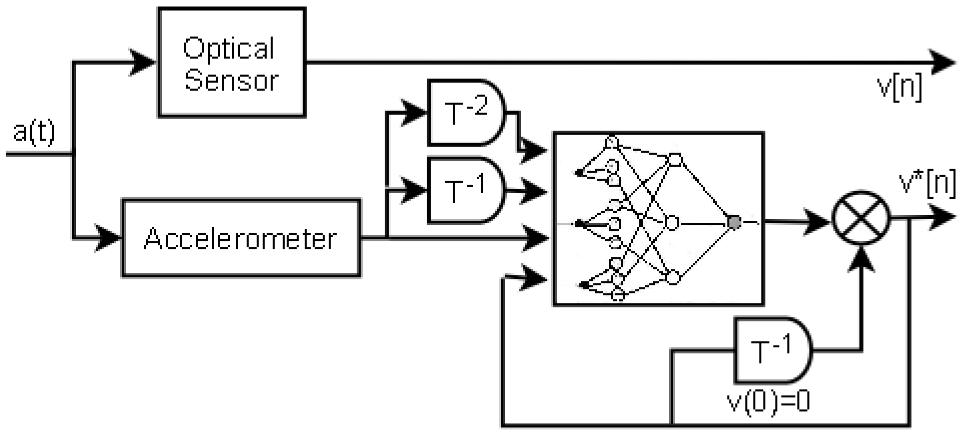


Figure 17: Diagram Block from Validation Phase

3.2.5 Fuzzy-Neural Model

The Fuzzy-Neural Network (FNN) proposed in this thesis consists to estimate the velocity of the prototype in one of X and Y axes. Therefore, the FNN has as output how much the current velocity should be incremented or decremented. The inputs of FNN are based on the measured acceleration and the current velocity for a specific axis.

The modeled Fuzzy-Neural Network has the same configuration commented in the section 3.2.4 , when explaining how the back-propagation algorithm works. For this particular configuration, all layers are quantified by counting the number of nodes, as observed in Table 2. Note that the number of rules is calculated based on number of inputs and the number of fuzzy sets of each input. The general formula is:

$$Number\ of\ Rules = \prod_{i=1}^{\substack{Number \\ of\ inputs}} \begin{pmatrix} Number\ of \\ Fuzzy\ Sets \\ of\ i^{th}\ input \end{pmatrix}$$

Layer 1 *Inputs*	Layer 2 *Membership Functions*	Layer 3 *Fuzzy Rules*	Layer 4 *Output*
Acceleration a[n]	POSITIVE A_0^p	$\mu_1 = A_0^p \cdot A_1^p \cdot A_2^p \cdot A_3^p$	
	NEGATIVE A_0^n	$\mu_2 = A_0^p \cdot A_1^p \cdot A_2^p \cdot A_3^n$	
	ZERO A_0^z	$\mu_3 = A_0^p \cdot A_1^p \cdot A_2^z \cdot A_3^p$	Velocity
Acceleration a[n-1]	POSITIVE A_1^p	$\mu_4 = A_0^p \cdot A_1^p \cdot A_2^z \cdot A_3^n$	Difference
	NEGATIVE A_1^n	$\mu_5 = A_0^p \cdot A_1^p \cdot A_2^n \cdot A_3^p$	$\Delta v = \frac{\sum_{i=1}^{N} \mu_i w_i}{\sum_{i=1}^{N} \mu_i}$
	ZERO A_1^z	$\vdots$	
Acceleration a[n-2]	POSITIVE A_2^p	$\mu_{50} = A_0^n \cdot A_1^n \cdot A_2^p \cdot A_3^z$	$+ \left(A_0^p - A_0^n\right) \cdot p$
	NEGATIVE A_2^n	$\mu_{51} = A_0^n \cdot A_1^n \cdot A_2^z \cdot A_3^p$	
	ZERO A_2^z	$\mu_{52} = A_0^n \cdot A_1^n \cdot A_2^z \cdot A_3^n$	
Current velocity v[n]	POSITIVE A_3^p	$\mu_{53} = A_0^n \cdot A_1^n \cdot A_2^n \cdot A_3^p$	
	NEGATIVE A_3^n	$\mu_{54} = A_0^n \cdot A_1^n \cdot A_2^n \cdot A_3^n$	
4 nodes	**11 nodes**	**54 nodes**	**1 node**

Table 2: Fuzzy-Neural Network table (first version)

Another thing that is possible to notice is that the fuzzy rules are symmetric. For example, the first rule μ_1 should have the same absolute value than last rule μ_{54}. That means:

If all input accelerations are positive and the current velocity is positive, the output should be positive with absolute value M.

If all input accelerations are negative and the current velocity is negative, the output should be negative with absolute value M.

The same relation can be done between μ_2 and μ_{53}, as well as the next symmetric pairs. This specific behavior is used to correct any asymmetry with the weights of the FNN that was calculated using back propagation algorithm, since each weight corresponds to the output of a fuzzy rule. Therefore, consider the trained weighting vector $W = [w_1\ w_2 \ldots w_N\ ]$, where N is the number of rules. The following equation can be used to correct the asymmetry between the weights.

$$W[i] = -W[N-i] = \frac{W[i]-W[N-i]}{2} \quad , \quad \text{where } i = \{0,1,\ldots,\frac{N}{2}\}$$

Another necessary modification on the weighting vector is when the input conditions have all accelerations equal ZERO. In this scenario, the FNN output should be equal ZERO. However, during the integration of the acceleration by using this FNN, it accumulates a lot of errors, which will cause a final velocity different from zero, even if the prototype is stationary. To avoid this behavior, a non-physical assumption is made when establishing the weighting vector:

"If the acceleration is nearly to ZERO, the velocity is also nearly to ZERO"

Translating this assumption in a singleton rule, the two following rules should not be influenced when training the FNN:

IF a_0 is *Zero* AND a_1 is *Zero* AND a_2 is *Zero* AND v is *Positive, THEN* $w = -M$

IF a_0 is *Zero* AND a_1 is *Zero* AND a_2 is *Zero* AND v is *Negative, THEN* $w = +M$

Where $a_0 = a[n]$, $a_1 = a[n-1]$, $a_2 = a[n-2]$, $v = v[n]$. Note that in case that all input accelerations are nearly ZERO, the FNN will "push" the velocity to ZERO as well.

This strategy to train and correct the weighting vector works well if the training data presents large acceleration. However, if the training data includes a significant number of acceleration close to ZERO, which is a common scenario when moving in low velocity, the training can deteriorate the final result of the FNN. For this reason, this algorithm does not work well for movements in low speed. In order to improve the performance of FNN for low speed, a second version of the FNN is created.

For better performance, the defuzzification process was modified by adding the output of the membership function of the current acceleration to the output signal. The modified Fuzzy-Neural Network is illustrated in the Figure 18. The weight *p* defines the intensity of the acceleration in the final output.

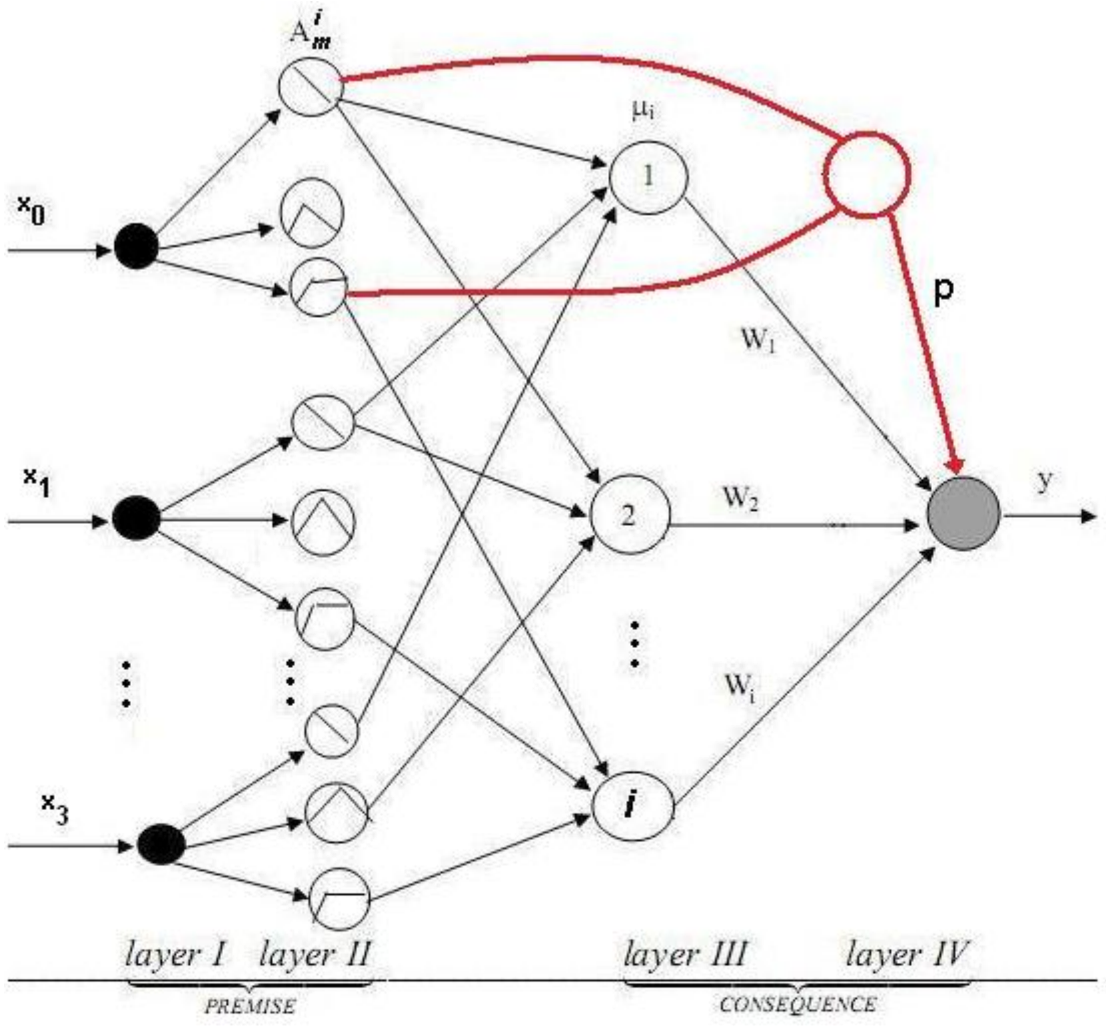


Figure 18: Modified Fuzzy-Neural Network

3.3 The Kalman Filter

The Kalman filter is a set of mathematical equations that provides an efficient computational means to estimate the state of a process, in a way that minimizes the mean of the squared error. The filter is very powerful in several aspects: it supports estimators of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown [10].

This filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman filter fall into two groups: *time update* equations and *measurement update* equations. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the *a priori* estimates for the next time step. The measurement update equations are responsible for the feedback.

The time update equations can also be thought of as *predictor* equations, while the measurement update equations can be thought of as *corrector* equations. Indeed the final estimation algorithm resembles that of a *predictor-corrector* algorithm for solving numerical problems as shown in Figure 19[10].

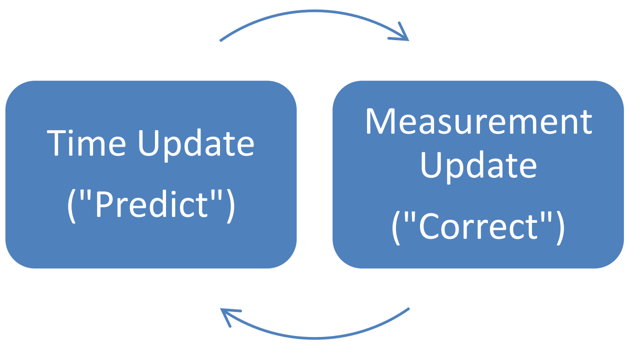


Figure 19: The discrete Kalman filter cycle

The Discrete Kalman filter time update equations are expressed as:

$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$ (Project the state ahead)

$P_k^- = AP_{k-1}A^T + Q$ (Project the error covariance ahead)

Where:

- $\hat{x}_k^-$ is the state estimation at time k
- A is the state transition model which is applied to the previous state $\hat{x}_{k-1}$
- B is the control-input model which is applied to the control vector u_k
- P_k^- is the estimate error covariance
- Q is the process noise covariance matrix

The Discrete Kalman Filter measurement update equations are:

$K_k = P_k^- H^T(HP_k^- H^T + R)^{-1}$ (Compute Kalman gain)

$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$ (Update estimate with measurement z_k)

$P_k = (I - K_k H)P_k^-$ (Update the error covariance)

Where:

- K_k is the Kalman gain
- H is the observation matrix
- R is the measurement noise covariance matrix
- z_k is the vector of measurements

3.3.1 The Process to be estimated

In this project, the Kalman filter is used to estimate the velocity based on the accelerations measurements. The position is not relevant here, once only the velocity must be sent to the host machine. Therefore, the state variables of Kalman filter are acceleration and velocity: $\hat{x}_k^- = \begin{bmatrix} v[k] \\ a[k] \end{bmatrix}$, where *v[k]* is current velocity and *a[k]* is the current acceleration.

The state transition model to represent the relation between the states can be defined base on the physical equation $v[k] = v[k-1] + a[k] \cdot dt$, therefore:

$$\hat{x}_k^- = \begin{bmatrix} v[k] \\ a[k] \end{bmatrix} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v[k-1] \\ a[k-1] \end{bmatrix}$$

The matrix *B*, the control-input model, is ignored for this application, since there is no control signal to be applied in this process. Note that dt represents the period time between two consecutives measurements.

The vector of measurements z_k includes the measurements from the accelerometer and from a tracking model of the velocity, which behaves as a virtual sensor. This tracking model it is also based in the same assumption made when using Fuzzy Neural Network. If the acceleration is nearly to zero, the velocity is also nearly to zero. Therefore, the following equations describe how the tracking model of the velocity works:

$$v_{TM}[k] = \begin{cases} 0, & ADa[k] < ADa_{min}\ AND\ z_{vib}[k] < z_{min} \\ v[k-1], & otherwise \end{cases}$$

Where $ADa[k]$ is the average deviation of the last N points multiplied by the current acceleration, and $z_{vib}[k]$ is the amplitude vibration in Z axis. The final vector of measurements is defined as: $z_k = \begin{bmatrix} v_{TM}[k] \\ a[k] \end{bmatrix}$. Since there are two measurements variables to be read, the observation matrix H is defined as $\begin{bmatrix} 1 & 0 \\ 0 & a_s \end{bmatrix}$, where a_s is the acceleration scale to convert the quantize value sampled using ADC to a specific scale that can be configure to behave similarly to the scale using optical sensor.

3.3.2 Filter Parameters and Tuning

In the actual implementation of the filter, the measurement noise covariance *R* is usually measured prior to operation of the filter. Measuring the measurement error covariance R is generally practical because the process needs to be measured anyway (while operating the filter), so it would be possible to take some off-line sample measurements in order to determine the variance of the measurement noise [10].

The determination of the process noise covariance Q is generally more difficult as typically is not possible to directly observe the process to be estimated. However, the covariance Q matrix can inject uncertainty into the process by selecting the elements from its diagonal. Consider $Q = \begin{bmatrix} Q_v & 0 \\ 0 & Q_a \end{bmatrix}$, where Q_v is related to the uncertainty of the velocity state and Q_a to the acceleration state. When moving in low speed with small acceleration, the signal coming from the accelerometers will be significantly small and the noise will have a large effect in the measurements. For this scenario, Q_v should have the same magnitude as Q_a. In the other scenario, when moving in high speed with large acceleration, the noise from the accelerometers will not interfere too much in the signal, so Q_v should be smaller than Q_a.

One way to dynamically change the matrix Q is to fix the value of Q_a and apply the following formula for Q_v:

$$Q_v[k] = \begin{cases} Q_v[k-1] \times 10.0, & if\ ADa[k] < ADa_{min}\ ,\ z_{vib}[k] < z_{min,}\ a[k] < a_{min} \\ Q_v[k-1] \times 0.10, & otherwise \end{cases}$$

However, Q_v should be limited in $10^{-5} \leq Q_v \leq Q_a$. Note that following points where the acceleration is really small and there is no large deviation, the magnitude of Q_v will be as large as Q_a, forcing the prediction of the velocity to zero, since the tracking model of the velocity is equal zero to this scenario. In case there is higher acceleration, the tracking model of the velocity will not affect too much the process, since Q_v will be smaller than Q_a for this scenario.

Using this approach, the matrix covariance Q changes dynamically, which will force the Kalman gain K_k to also change dynamically. Therefore, it will not be possible to pre-compute this parameter by running the filter off-line.

3.4 The State-Machine Estimator

The Fuzzy Neural Network and Kalman-Filter solutions proposed in this thesis are constantly integrating the acceleration to estimate the velocity. However, this integration may change for different situations, like when moving with high acceleration and low acceleration. A general formula to represent the integration of the acceleration can be expressed as:

$$v(t) = \int_{t_0}^{t_f} [a(t) + a_{bias}(t) + a_{noise}(t)] \cdot dt$$

The signal coming from the accelerometers has a bias DC component and noise. The bias DC component can be almost totally removed using the bias filter presented in the section 3.1.2 . The noise component can be reduced using the benefits of Kalman Filter and/or Fuzzy-Neural Network. However, the biggest problem that injects uncertainty in the process is the continuous integration, which accumulates error during the movement. This error is proportional to the time of integration ($\Delta t = t_f - t_0$). Therefore, one way to solve this problem is to break the movement in small parts and ignore the integration of the acceleration after Δt is too large. One way to limit the time of integration is to establish the following rule:

After integrating the first points and the module of velocity assumes a value higher than a specific minimum threshold, the integration of the acceleration should continue until the velocity crosses to zero.

The meaning of this rule is that the user can only move the mouse to one direction. If it tries to change the direction, the acceleration is ignored and the velocity is kept to zero until the movement is finished. Therefore, the user will not be allowed to change the direction if he does not stop moving the mouse.

Description of State-Machines

State-Machines can be described as a model of behavior composed of a finite number of states, transitions between those states, and actions. For identifying the different behaviors when moving a mouse device, a finite number of states are designed to distinguish the different parts of a movement, especially to determine when the mouse device is not moving，accelerating and decelerating. Different state-machines are proposed to integrate the acceleration. Each one will be explained further in the following sections.

3.4.1 State-Machine using simple integration method

If a state machine is designed to describe the movement of a mouse device, the most obvious and initial state is when the user is not moving the mouse device. This state will be called IDLE state, when the acceleration signal is basically constituted by the intrinsic noise from the accelerometers. When a peak of acceleration is detected, a transition is occurred, changing the state from IDLE to another state called ACCELERATING state. In this state, the acceleration signal is integrated for estimating the velocity. This integration can be really simple, when integrating directly the acceleration signal, or applying some more complex technique, like Kalman Filter. For this initial state-machine, a simple integration of acceleration combined with the trapezoidal method to reduce the error of integration is used. The estimated instantaneous velocity can be obtained by summing the areas between two following sampled accelerometer signals. As observed in the Figure 20.

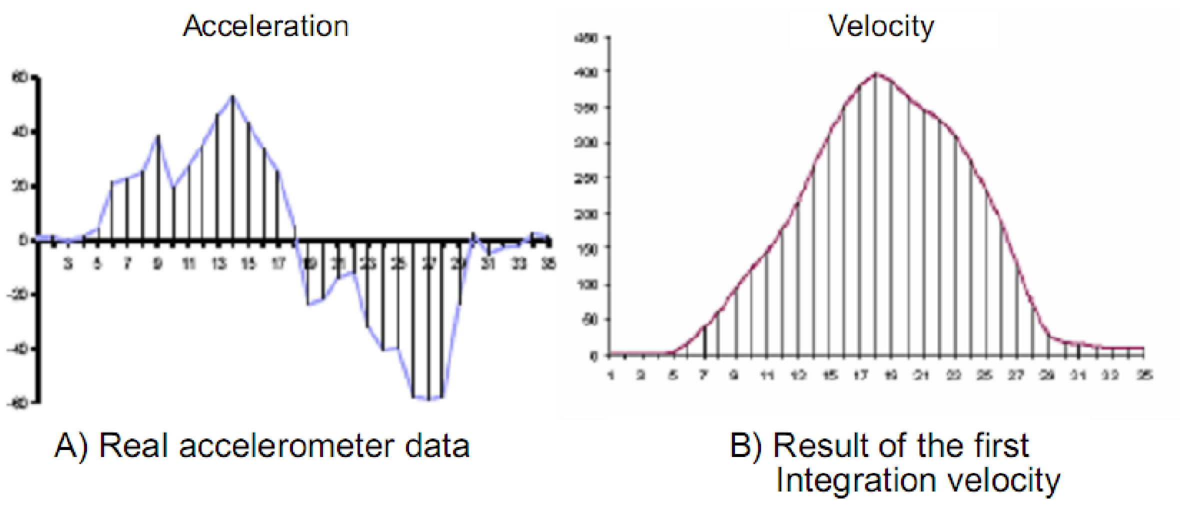


Figure 20: Approximation of the Instantaneous Velocity

The trapezoid method commented previously consider the area between two sampled acceleration as a trapezoid, instead of a rectangle. The final formula of this area is represented as:

$$Area[k] = \left[a[k] + \frac{(a[k] - a[k-1])}{2}\right] \cdot \Delta t$$

Where Δt is the period between two sampled acceleration signals. Finally, the instantaneous velocity is represented as:

$$v[k] = Area[k] + v[k-1]$$

The above equation should be applied when the current state is the ACCELERATING state. A transition will only occur after the acceleration is integrated a minimum number of times and some deceleration is detected. In this scenario, there are two possible states to go. The first one is called HIGH SPEED state, and the second one is LOW SPEED state. The decision is made by observing the current velocity at the moment the transition occurred. If it is higher than a pre-specified threshold, the next state is the HIGH SPEED state, if not, the LOW SPEED state.

For the HIGH SPEED state, the same equation to calculate the instantaneous velocity is used. However, if the last points of the acceleration are too small, instead of integrating the acceleration, the current velocity is decremented. The reason to apply this strategy is because in case there is any accumulated error of integration after the movement is finished, the velocity will return to ZERO after some cycles.

For the LOW SPEED state, the current velocity also is decremented if the last points of the acceleration are too small. However, for any other situation, the velocity will be kept constant, since when moving the mouse device with low speed, the acceleration signal also is small, which would prejudice the estimation of the velocity.

Once the velocity crosses to zero, a transition occurs, changing from the previous state to STABILIZING state. In this state, the velocity is kept in ZERO until the last points of acceleration are small, which will probably indicate that the user stop moving the device. Once non-movement is detected, the next state will return to IDLE state. The complete state machine is illustrated in Figure 21.

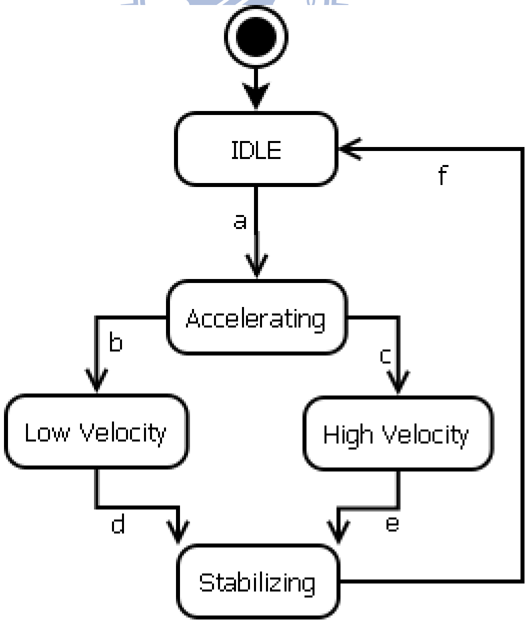


Figure 21: Simple State-Machine

A table with all states, transitions and actions is shown in the Table 3. Note that each axis (X and Y) will use this state machine, therefore the actions are not combined and each state machine may have different current states.

IDLE state	
Action	$v[k] = 0$
Transitions	**a:** peak detected in the acceleration signal
ACCELERATING state	
Action	$v[k] = v[k-1] + \left[a[k] + \frac{(a[k] - a[k-1])}{2}\right] \cdot \Delta t$
Transitions	**b:** deceleration detected and current velocity small **c:** deceleration detected and current velocity high
LOW VELOCITY state	
Action	$IF\ ADa[k] < ADa_{min}\ \ AND\ \ Amp_z < Amp_{min}$ $v[k] = v[k-1] - SIGN(v[k-1]) \cdot D$ ELSE $v[k] = v[k-1]$
Transitions	**d:** current velocity crosses to zero
HIGH VELOCITY state	
Action	$IF\ ADa[k] < ADa_{min}\ \ AND\ \ Amp_z < Amp_{min}$ $v[k] = v[k-1] - SIGN(v[k-1]) \cdot D$ ELSE $v[k] = v[k-1] + \left[a[k] + \frac{(a[k] - a[k-1])}{2}\right] \cdot \Delta t$
Transitions	**e:** current velocity crosses to zero
STABILIZING state	
Action	$v[k] = 0$
Transitions	**f:** average deviation combined with current acceleration is small

Table 3: States, transitions and actions of the State-Machine using simple integration

Note that in the formulas from LOW/HIGH VELOCITY states, the element D represents how much the current velocity is decremented.

3.4.2 State-Machine using Kalman filter

In this solution, the same state-machine from the previous section is kept. However, the actions are different. In this case, the Kalman filter equations are used to integrate the acceleration signal. The parameters used for this Kalman Filter are slightly different from the ones presented in section 3.3 . For instance, the process noise covariance matrix Q is constant, with $Q_v = Q_a$. Therefore, the Kalman gain will be also a constant after the transition phase. In this way, it is only necessary to feed the measurements with the current acceleration and the tracking model of velocity. The predicted velocity from the Kalman filter may be used to define the velocity in the mouse device. For simplifying the equations when building the action table, consider the following formula to represent the Kalman Filter:

$$v[k] = Kalman(z_k) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \hat{x}_k^-$$

Where:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

The transitions of this state-machine are the same than the previous one. Only the actions are different. Observe that even when the prediction of the Kalman Filter is not used, the function must be called to update the states of the filter. All states, transitions and actions are described in Table 4.

IDLE state	
Action	$v[k] = 0$ $Kalman\left(\begin{bmatrix} 0 \\ a[k] \end{bmatrix}\right)$
Transitions	**a:** peak detected in the acceleration signal
ACCELERATING state	
Action	$v[k] = Kalman\left(\begin{bmatrix} v[k-1] \\ a[k] \end{bmatrix}\right)$
Transitions	**b:** deceleration detected and current velocity small **c:** deceleration detected and current velocity high
LOW VELOCITY state	
Action	$IF\ ADa[k] < ADa_{min}\ \ AND\ \ Amp_z < Amp_{min}$ $v[k] = Kalman\left(\begin{bmatrix} 0 \\ a[k] \end{bmatrix}\right)$ ELSE $v[k] = Kalman\left(\begin{bmatrix} v[k-1] \\ 0 \end{bmatrix}\right)$
Transitions	**d:** current velocity crosses to zero
HIGH VELOCITY state	
Action	$IF\ ADa[k] < ADa_{min}\ \ AND\ \ Amp_z < Amp_{min}$ $v[k] = Kalman\left(\begin{bmatrix} 0 \\ a[k] \end{bmatrix}\right)$ ELSE $v[k] = Kalman\left(\begin{bmatrix} v[k-1] \\ a[k] \end{bmatrix}\right)$
Transitions	**e:** current velocity crosses to zero
STABILIZING state	
Action	$v[k] = 0$ $Kalman\left(\begin{bmatrix} 0 \\ a[k] \end{bmatrix}\right)$
Transitions	**f:** average deviation combined with current acceleration is small

Table 4: States, transitions and actions of the State-Machine using Kalman Filter

3.4.3 Combined Axis XY State-Machine

Instead of using two state-machines to deal with X and Y axis, a single state-machine is proposed, which combines both axes when defining the transitions and actions of the state-machine. Since the combination of two directions is analyzed in this solution, it is possible to define what direction the user is moving the mouse. For example, to RIGHT or LEFT, BACKWARD or FORWARD and DIAGONAL. For each combination, a state is defined in this state-machine. Also, there are more states to classify if the movement is in low speed or high speed. This state-machine is illustrated in the Figure 22.

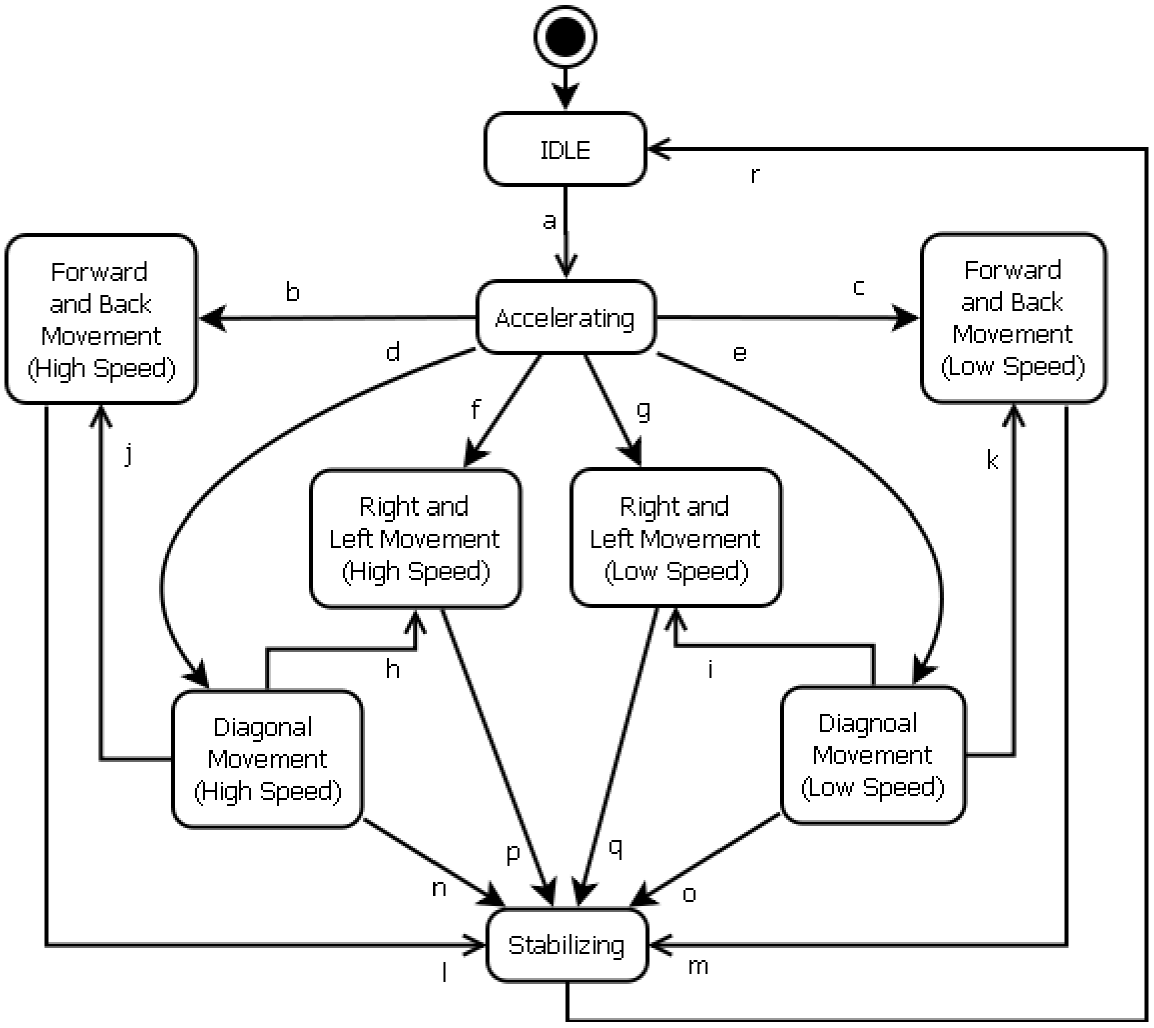

IDLE
r
a
Forward and Back Movement (High Speed)
b
Accelerating
c
Forward and Back Movement (Low Speed)
d
e
f
g
j
k
Right and Left Movement (High Speed)
Right and Left Movement (Low Speed)
h
i
Diagonal Movement (High Speed)
Diagnoal Movement (Low Speed)
p
q
n
o
Stabilizing
l
m

Figure 22: Combined State-Machine

All states, transitions and actions are described in the following table. Note that the equations from this solution are the same as the solution using the simple state-machine; however, one state has actions to both axes at the same time. Moreover, the transitions demand the information of both axes.

	IDLE state
Action	$v_x[k] = 0$ $v_y[k] = 0$
Transitions	**a:** peak detected in the acceleration signal from X or Y
	ACCELERATING state
Action	$v_x[k] = v_x[k-1] + \left[a_x[k] + \frac{(a_x[k] - a_x[k-1])}{2}\right] \cdot \Delta t$ $v_y[k] = v_y[k-1] + \left[a_y[k] + \frac{(a_y[k] - a_y[k-1])}{2}\right] \cdot \Delta t$
Transitions	**b:** deceleration detected in X&Y and $v_y \gg v_x$ and v_y is high **c:** deceleration detected in X&Y and $v_y \gg v_x$ and v_y is small **d:** deceleration detected in X&Y and $v_x \approx v_y$ and v_{xy} is high **e:** deceleration detected in X&Y and $v_x \approx v_y$ and v_{xy} is small **f:** deceleration detected in X&Y and $v_x \gg v_y$ and v_x is high **g:** deceleration detected in X&Y and $v_x \gg v_y$ and v_x is small
	FORWARD and BACKWARD MOVEMENT (high speed) state
Action	$IF\ ADa_y[k] < ADa_{min}\ \ AND\ \ Amp_z < Amp_{min}$ $\quad v_y[k] = v_y[k-1] - SIGN(v_y[k-1]) \cdot D$ $\quad v_x[k] = 0$ ELSE $\quad v_y[k] = v_y[k-1] + \left[a_y[k] + \frac{(a_y[k] - a_y[k-1])}{2}\right] \cdot \Delta t$ $\quad v_x[k] = 0$
Transitions	**l:** v_y crosses to zero

FORWARD and BACKWARD MOVEMENT (low speed) state	
Action	$IF\ ADa_y[k] < ADa_{min}\ \ AND\ \ Amp_z < Amp_{min}$ $v_y[k] = v_y[k-1] - SIGN(v_y[k-1]) \cdot D$ $v_x[k] = 0$ ELSE $v_y[k] = v_y[k-1]$ $v_x[k] = 0$
Transitions	**m:** v_y crosses to zero
DIAGONAL MOVEMENT (high speed) state	
Action	$IF\ ADa_{xy}[k] < ADa_{min}\ \ AND\ \ Amp_z < Amp_{min}$ $v_y[k] = v_y[k-1] - SIGN(v_y[k-1]) \cdot D$ $v_x[k] = v_x[k-1] - SIGN(v_x[k-1]) \cdot D$ ELSE $v_y[k] = v_y[k-1] + \left[a_y[k] + \frac{(a_y[k] - a_y[k-1])}{2}\right] \cdot \Delta t$ $v_x[k] = v_x[k-1] + \left[a_x[k] + \frac{(a_x[k] - a_x[k-1])}{2}\right] \cdot \Delta t$
Transitions	**h:** v_y crosses to zero **j:** v_x crosses to zero **n:** v_{xy} crosses to zero
DIAGONAL MOVEMENT (low speed) state	
Action	$IF\ ADa_{xy}[k] < ADa_{min}\ \ AND\ \ Amp_z < Amp_{min}$ $v_y[k] = v_y[k-1] - SIGN(v_y[k-1]) \cdot D$ $v_x[k] = v_x[k-1] - SIGN(v_x[k-1]) \cdot D$ ELSE $v_y[k] = v_y[k-1]$ $v_x[k] = v_x[k-1]$
Transitions	**i:** v_y crosses to zero **k:** v_x crosses to zero **o:** v_{xy} crosses to zero

RIGHT and LEFT MOVEMENT (high speed) state	
Action	$IF\ ADa_x[k] < ADa_{min}\ \ AND\ \ Amp_z < Amp_{min}$ $v_y[k] = 0$ $v_x[k] = v_x[k-1] - SIGN(v_x[k-1]) \cdot D$ ELSE $v_y[k] = 0$ $v_x[k] = v_x[k-1] + \left[a_x[k] + \frac{(a_x[k] - a_x[k-1])}{2}\right] \cdot \Delta t$
Transitions	**p:** v_x crosses to zero
RIGHT and LEFT MOVEMENT (low speed) state	
Action	$IF\ ADa_x[k] < ADa_{min}\ \ AND\ \ Amp_z < Amp_{min}$ $v_y[k] = 0$ $v_x[k] = v_x[k-1] - SIGN(v_x[k-1]) \cdot D$ ELSE $v_y[k] = 0$ $v_x[k] = v_x[k-1]$
Transitions	**q:** v_x crosses to zero
STABILIZING state	
Action	$v_x[k] = 0$ $v_y[k] = 0$
Transitions	**r:** average deviation combined with current acceleration in X and Y are smaller than a pre-defined threshold

Table 5: States, transitions and actions of the Combined State-Machine

3.4.4 Combined State-Machine using motion detection sensor

One of the most difficult problems to solve when using mouse devices based on accelerometers is to determine if the device is moving or not. Using only accelerometers, this condition can be checked observing the average deviation of the signal and the magnitude of the acceleration from the last points. If both are smaller than a pre-defined threshold, there is a higher probability that the device is stationary. However, this scenario can be misinterpreted when the device is moving in constant and low velocity, which will induce a small acceleration that is undetectable. To avoid this misinterpretation, a second sensor is proposed. Considering the device has a binary sensor that detects if there is movement or not, how much the performance can be improved?

To answer this question, the same state-machine from the previous solution (Figure 22) is used; however the following transitions are modified:

- Transition **(a):** if the motion detection sensor detects movement;
- Transitions **(l,m,n,o,p,q,r):** the conditions from the previous solution OR the motion detection sensor detects no movement.

The current prototype was not designed to have the motion detection sensor. However, the optical sensor can be used to emulate the behavior of this sensor; therefore it will be possible to analyze the performance improvement adding such sensor.

Another sensor that was considered as a substitute for optical sensor as motion detection sensor was a digital microphone. In this case, the microphone is attached to the base plate, which let the white noise generated by the friction between the base plate and the surface be captured by the microphone. However, since the prototype has only a limited microcontroller, it is not possible to simply attach a microphone to the current prototype. To overcome this limitation, a dedicated embedded platform to process the signal from the microphone is used, which is based on a FPGA. Therefore, a filter can be implemented in hardware to convert the wave signal to a binary sensor that follows the guidelines already explained in this section. The output of this filter is assigned to an output pin in the dedicated embedded platform, which is wire connected to an input pin in the current prototype.

3.4.5 Combined State-Machine using physical button

Following the same reasoning from the previous solution, another “sensor” is proposed, however in this solution, this sensor is substituted by a physical button, which can easily be attached to the current prototype. In this way, the user can have a better control of the mouse device, pressing the button when it is desired to move the cursor on screen. For this case, the same state-machine is also used; however, the transitions are modified as follow:

- Transition **(a):** if the button is pushed AND a peak is detected in X or Y;
- Transitions **(l,m,n,o,p,q,r):** the conditions from the previous solution OR the button is released.

3.4.6 State-Machine using inertial behavior

In this solution, a totally new approach is proposed to move the mouse device. This approach is based on the inertial behavior of an object when accelerated to any direction. The initial acceleration on the object originates from the force applied on it. Once the force is suspended, the object will still keep moving until the friction force, which opposes to the direction of the movement, annuls the motion. The equations that describe the velocity during the movement can be split in two parts:

The first part represents the moment that the object is under an external force, which is describe as:

$$\vec{F}_{ext} = m \cdot \vec{a}$$

Where m is the mass of the object and $\vec{a}$ is the acceleration originated by applying the external force $\vec{F}_{ext}$. The velocity can be obtained by integrating this acceleration:

$$v(t) = \int_{t_0}^{t_f} a(t) \cdot dt$$

The second part represents when the object is under the friction force, however, with some accumulated kinetic energy, which is so called in this project of the free movement moment. The equations that describe the velocity in this part can be obtained by using the conservation of energy equation, which says:

$$E_k + E_f = C$$

Where E_k is the kinetic energy, E_f is the energy dissipated because of the friction and C is a constant. The kinetic energy is represented as $E_k = \frac{1}{2}mv^2$, and the friction energy is represented as $E_f = \mu \cdot mg \cdot s$, where μ is the friction coeficient, g is the gravity and s is the distanced traveled. Deriving the conservation of energy equation, it is obtained:

$$m \cdot v(t) \cdot \frac{v(t)}{dt} + \mu \cdot mg \cdot v(t) = 0$$

Integrating the equation above, and considering that the initial velocity is the final velocity of the first part $v(t_f)$, the following equation is obtained:

$$v(t) = v\left(t_f\right) - \mu \cdot g \cdot t$$

This indicates that the velocity will be reduced linearly during the free movement moment. Now, applying a similar behavior to the way the user can operate a mouse device, and designed a state-machine to represent it, it is possible to define a state that explains the initial acceleration, which is quite similar to the state ACCELERATION from the other state-machines. However, once a deceleration is detected, the following acceleration samples should be ignored, since the idea of the free movement is decrement the velocity along the time, emulating an imaginary friction force. The number of samples to be ignored should be proportional to the number of samples integrated in the ACCELERATION state. Plus, the average deviation should be checked to confirm if the user stopped moving the mouse. Note that in this solution, the user only needs to perform short movements, and only the first samples of the acceleration will be integrated, defining the maximum velocity for that movement. After that, the velocity will be decremented. From the physical equations that explain the movement of the object, the velocity is supposed to decrement linearly. However, when using as mouse device, it will better work if the velocity is decremented exponentially, because the cursor velocity reaches zero faster, letting the user better control the cursor on screen.

Another feature that the user can perform to increase even more the control to the cursor on screen is to have an option to stop the cursor. This can be made by simply taping or hitting the prototype. Every time the user applies any force in the Z direction, a significant increase of the vibration in the Z axis accelerometer is detected. The final state-machine containing all these actions and transitions are illustrated in Figure 23.

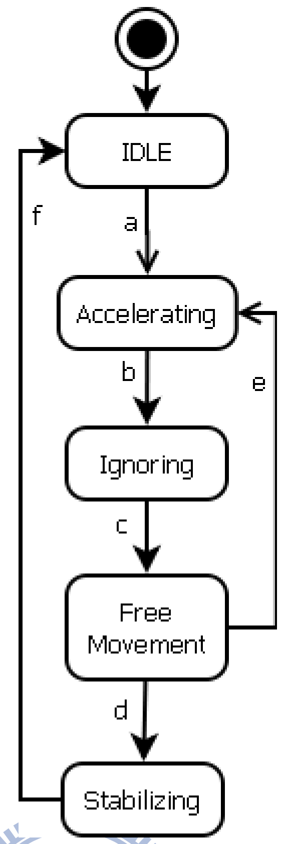


Figure 23: Free Movement State-Machine

All states, transitions and actions are described in the following table.

IDLE state	
Action	$v_x[k] = 0$ $v_y[k] = 0$
Transitions	**a:** peak detected in the acceleration signal of X or Y
ACCELERATING state	
Action	$v_x[k] = v_x[k-1] + \left[a_x[k] + \frac{(a_x[k] - a_x[k-1])}{2}\right] \cdot \Delta t$ $v_y[k] = v_y[k-1] + \left[a_y[k] + \frac{(a_y[k] - a_y[k-1])}{2}\right] \cdot \Delta t$
Transitions	**b:** deceleration detected in X and Y

IGNORING state	
Action	$v_x[k] = v_x[k-1]$ $v_y[k] = v_y[k-1]$
Transitions	**c:** counter limited reached and small average deviation in X and Y
FREE MOVEMENT state	
Action	$v_x[k] = v_x[k-1] \cdot \alpha$ $v_y[k] = v_y[k-1] \cdot \alpha$
Transitions	**d:** Huge vibration in Z and small amplitude in X and Y **e**: Peak detected in X or Y
STABILIZING state	
Action	$v_x[k] = 0$ $v_y[k] = 0$
Transitions	**f:** counter limited reached

Table 6: States, transitions and actions of the Free Movement State-Machine

Note that in the action of the FREE MOVEMENT state, α is a number slightly smaller than 1, and will define how fast the velocity will converge to zero. It is also possible to notice that the user can change the direction of the velocity in both axes when the current state is in the FREE MOVEMENT state.

Chapter 4. Testing and Experimental Results

4.1 Analytical Results

All techniques explained in the previous chapter will be compared in this section, having as reference the velocity curve generated by the optical sensor measurements in X and Y. The curves are generated by moving the prototype in specific scenarios, like low average speed movement, high average speed movement and sinusoidal movement. The data generated by reading the optical sensors and accelerometers are sent to the host machine. After the collecting the data, the velocity in both axes X and Y will be estimated using different techniques, and graphics will be generated and some mathematical indicators will be extract from the estimated velocities.

The units in the graphics are based in the format data that has to be sent using the HID device class (from USB driver). The values of velocity can change from -128 to 127, which represents one signed byte. The X coordinates from the graphics represented the number of cycles, where each cycle has a period of 8 milliseconds (the maximum period when sending packets using HID device class).

4.1.1 Graphical Results

Fuzzy-Neural Network with and without Training

The differences between a Fuzzy-Neural Network without training and with training are illustrated in the Figure 24. After training, the velocity curve of the fuzzy-neural network avoids the reverse velocity case for this specific scenario (only one axis is analyzed).

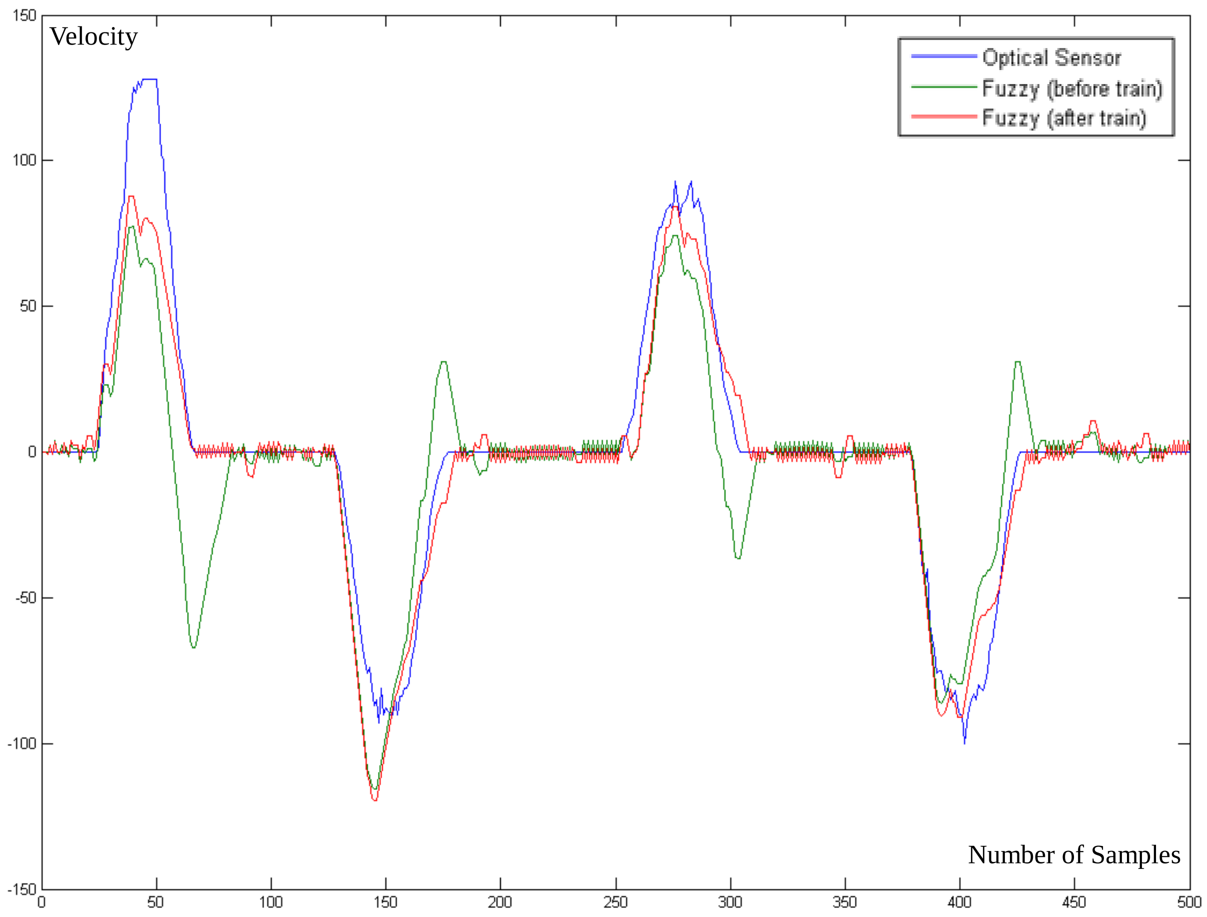


Figure 24: Fuzzy-Neural Network with and without training

Signal Processing Methods Comparison

Three techniques are compared to each other and to the reference velocity from the optical sensor. Three scenarios are used to analyze the performance of each technique – high speed, low speed and sinusoidal scenarios.

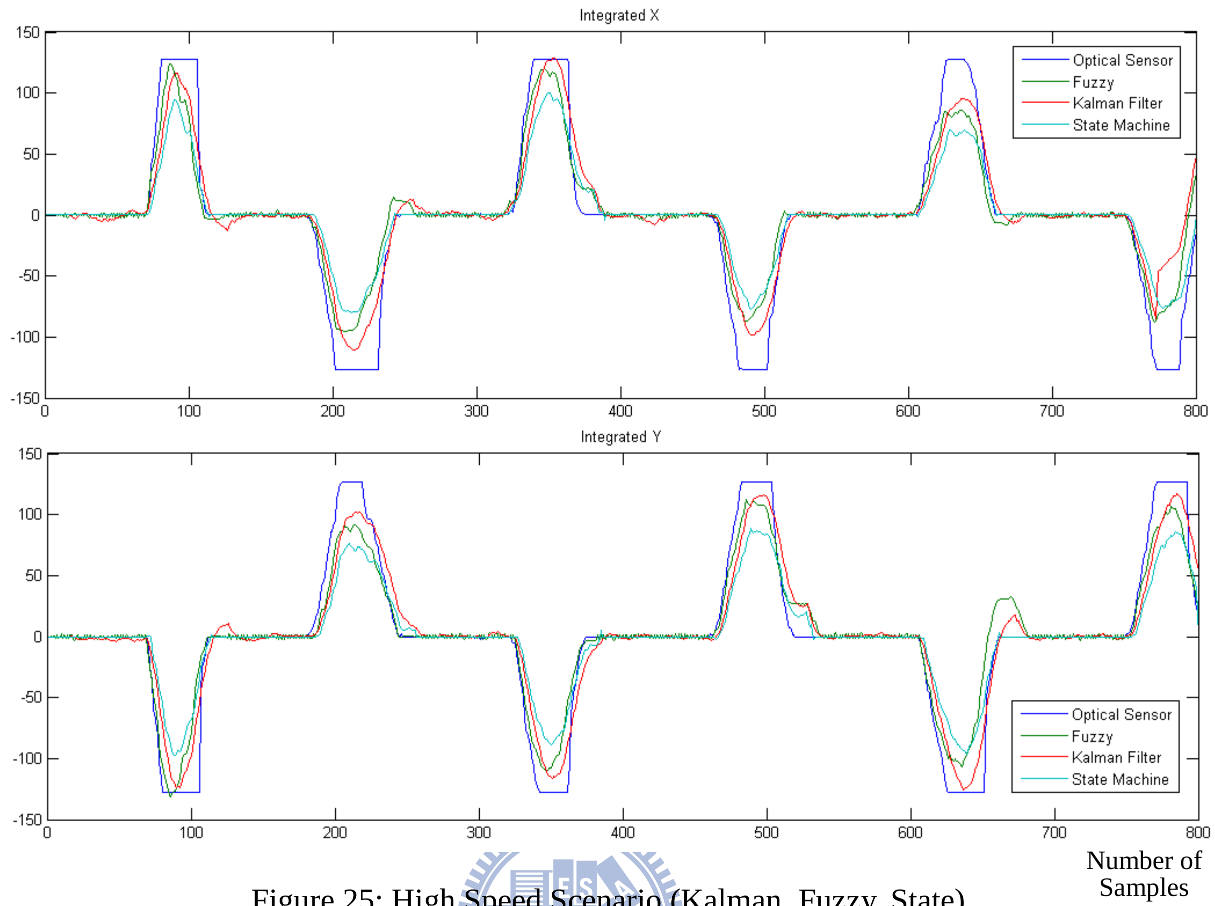


Figure 25: High Speed Scenario (Kalman, Fuzzy, State)

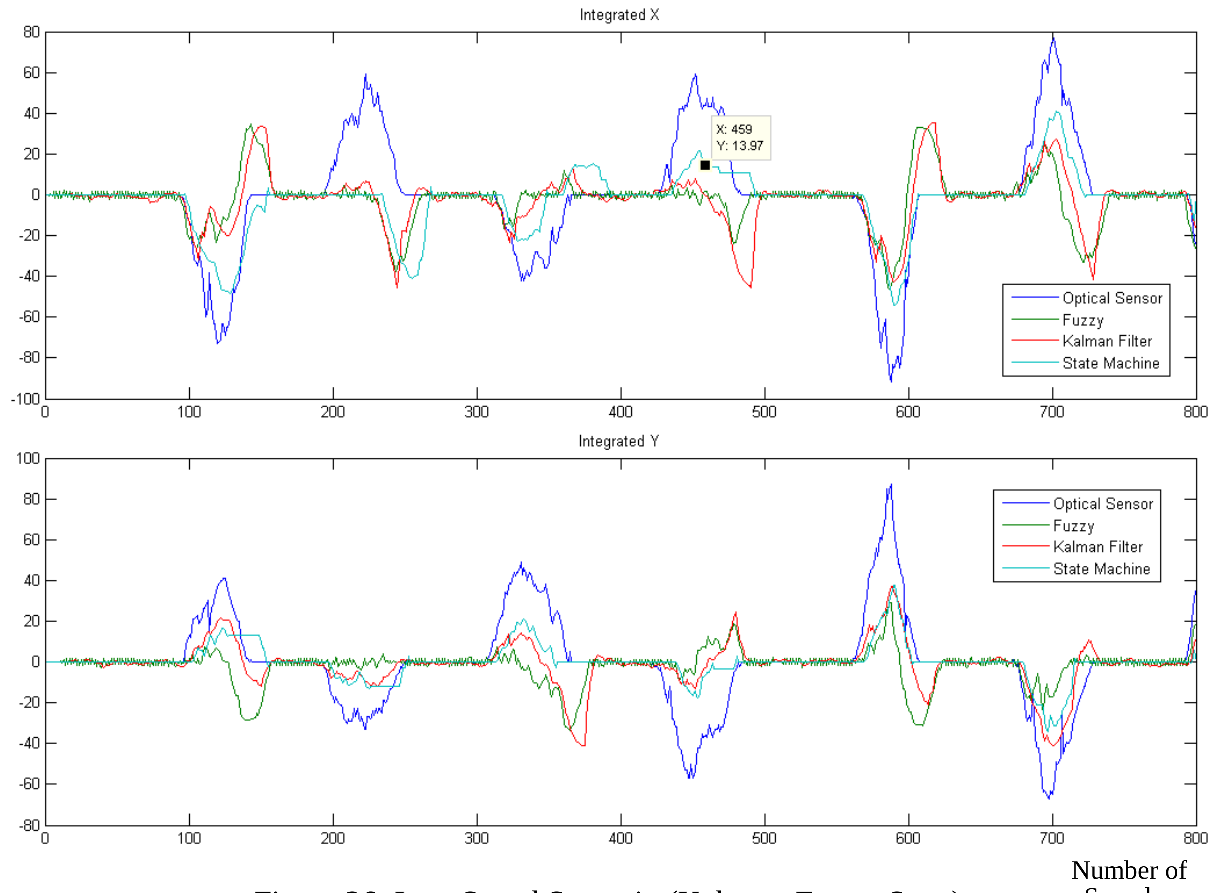


Figure 26: Low Speed Scenario (Kalman, Fuzzy, State)

In the high speed scenario (Figure 25), it is possible to notice that sometimes exist a residual velocity after stop moving the mouse device. This residual velocity is really prejudicial when applying to a mouse device, which can let the user lose control of the cursor on the screen. The worst case is when this residual velocity has different signal from the predominant direction of the movement, which will cause a reverse velocity in the end of the movement. This behavior is quite more serious in the low speed scenario (Figure 26), where most of the tests show a significant residual velocity. From the solutions analyzed, it is possible to notice that the State-Machine estimator works really well in high speed scenario, avoiding a negative residual velocity. In low speed scenario, not all cases had a good performance.

The biggest problem when working in low speed scenario is that sometimes is not possible to detect any peak of acceleration, because the real acceleration is occulted by the noise from the accelerometer. An example is illustrated in the Figure 27.

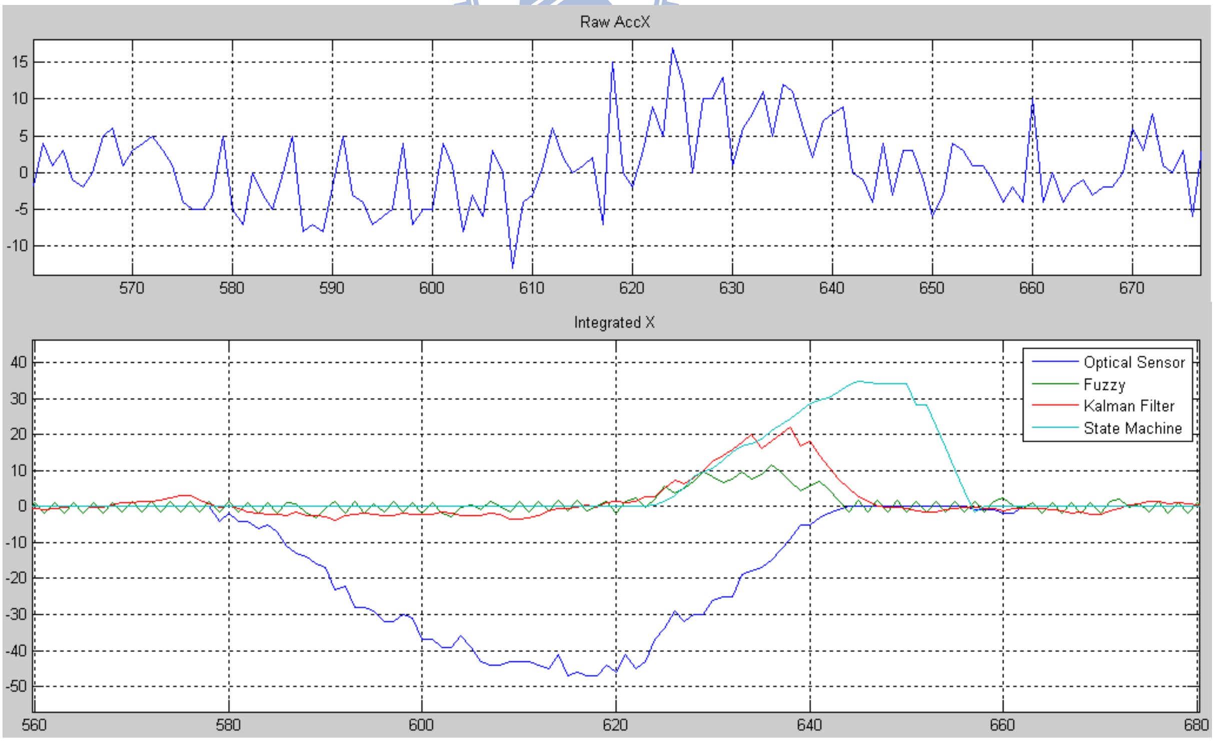


Figure 27: No peak detection when moving in low speed

The first graphic shows the acceleration signal from X axis and the second graphic shows the estimated velocity. It is possible to notice that the movement starts around the 580th cycle, however if the acceleration signal is observed, there is no evidence of any peak of acceleration accusing some movement.

Another problem is when moving long movements, especially when changing the direction of the velocity, such as sinusoidal movements. In this scenario (Figure 28), no technique can work correctly. Kalman-filter and Fuzzy-Neural estimators accumulate huge amount of error during the integration of the acceleration. And the State-Machine estimator simply would ignore most of the movement, only integrating the first points, until the velocity crosses to zero.

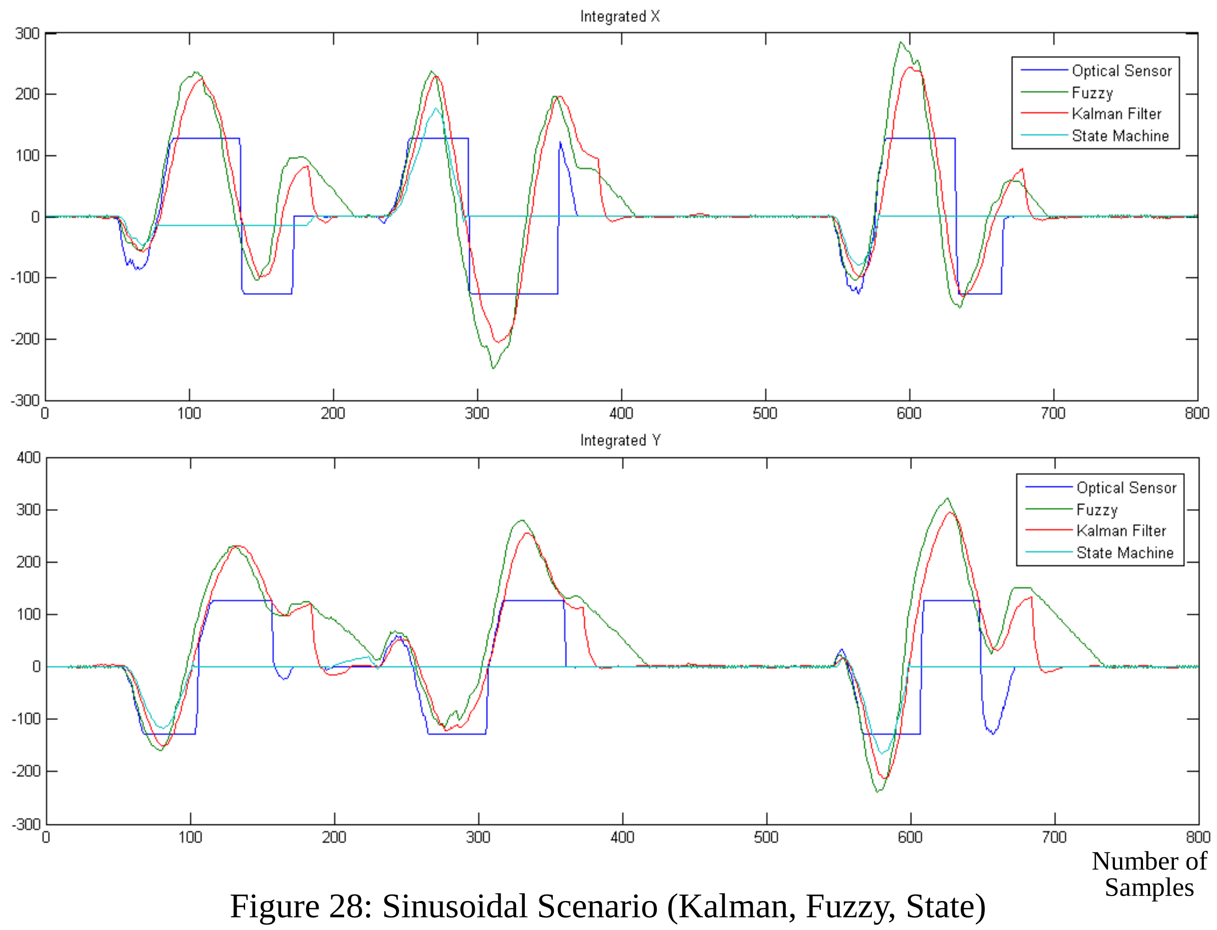


Figure 28: Sinusoidal Scenario (Kalman, Fuzzy, State)

Simple Integration and Kalman Filter Comparison

Both techniques based on the same state-machine are compared. The unique difference between them is the way the acceleration is integrated. The first solution uses a simple integration method and the second one use Kalman filter.

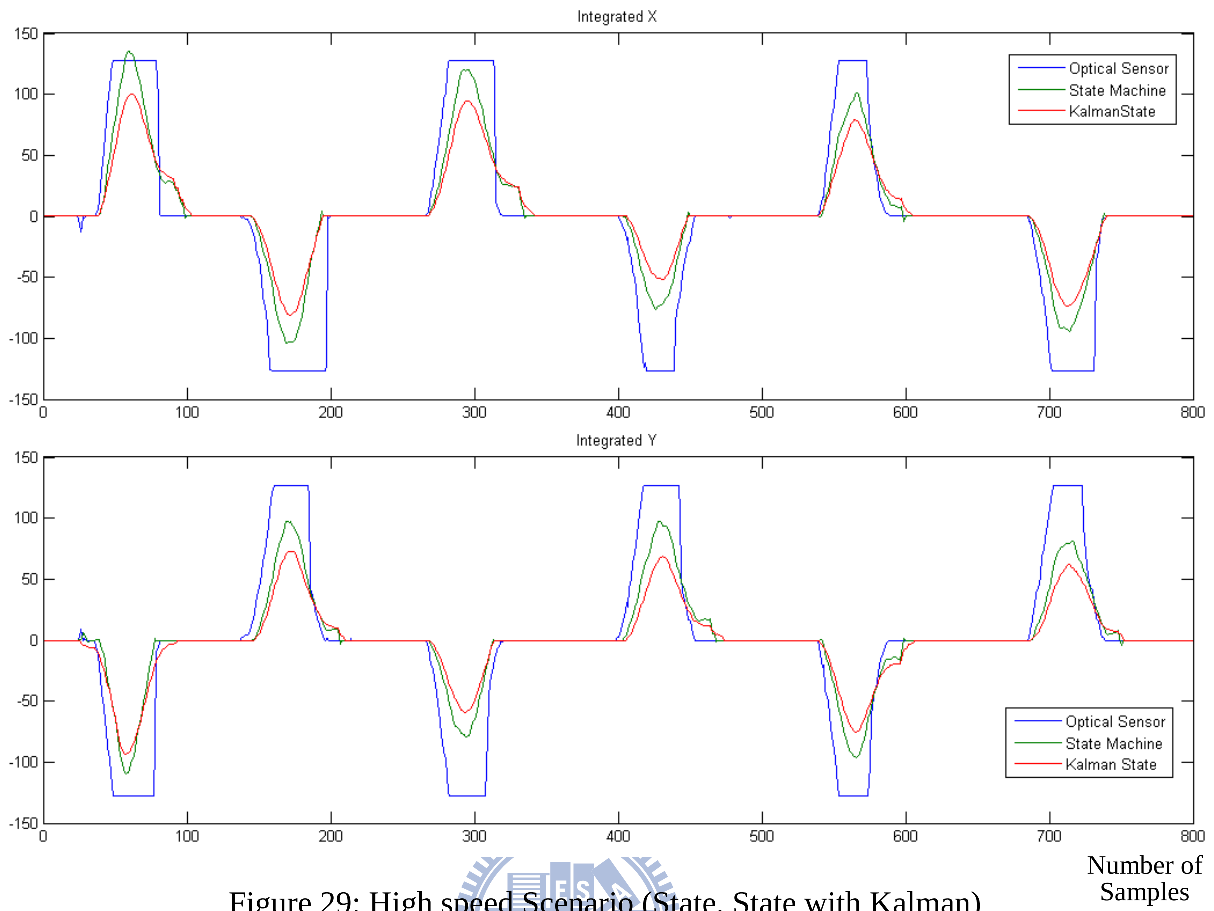


Figure 29: High speed Scenario (State, State with Kalman)

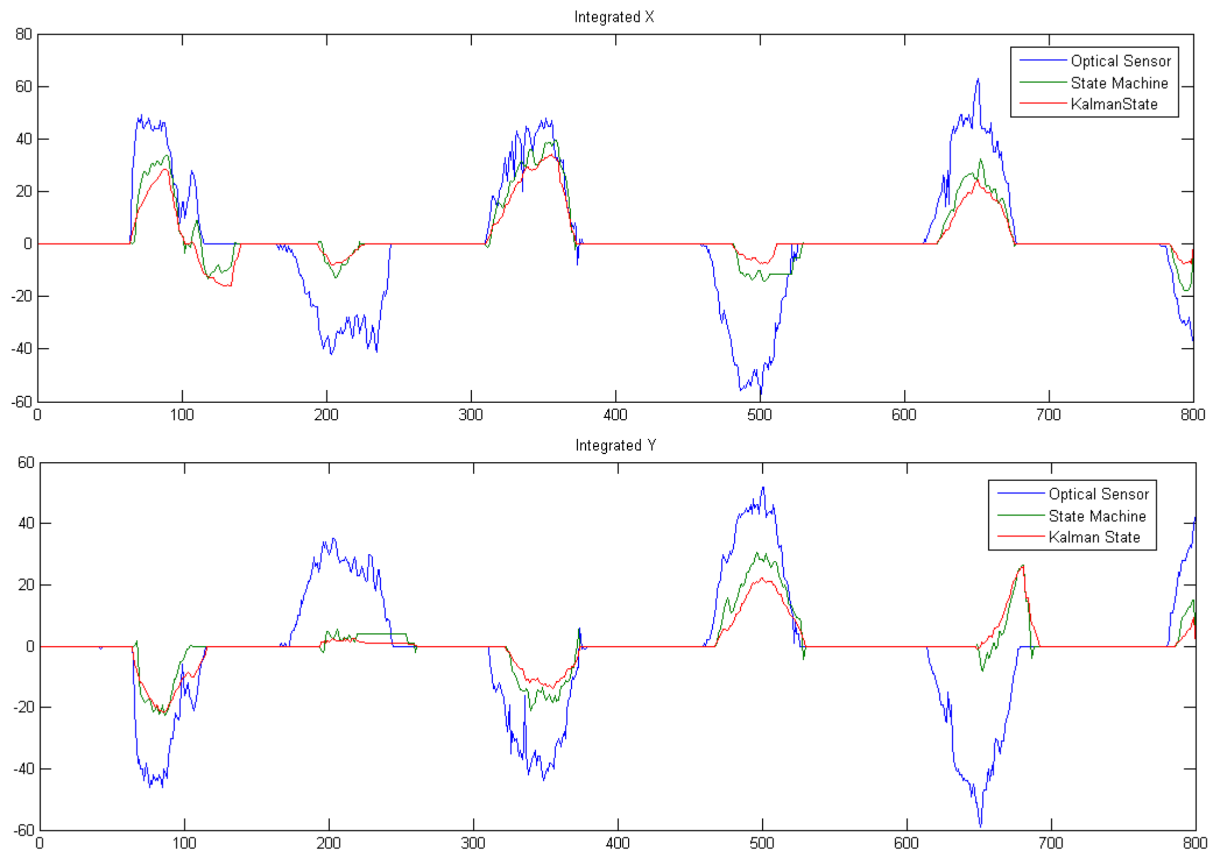


Figure 30: Low speed Scenario (State, State with Kalman)

Observing both graphics from Figure 29 and Figure 30, their behaviors are similar, presenting the same problems of negative and positive residual velocities. However, it would be easier to port the algorithm from the state-machine using simple integration to the microcontroller from the prototype. It would also consumes less resources (memory and processor), since there is no significant advantage using Kalman filter for these scenarios.

State-Machines Comparison

In this case, the advantage to have a unique state-machine that combines both axes is presented.

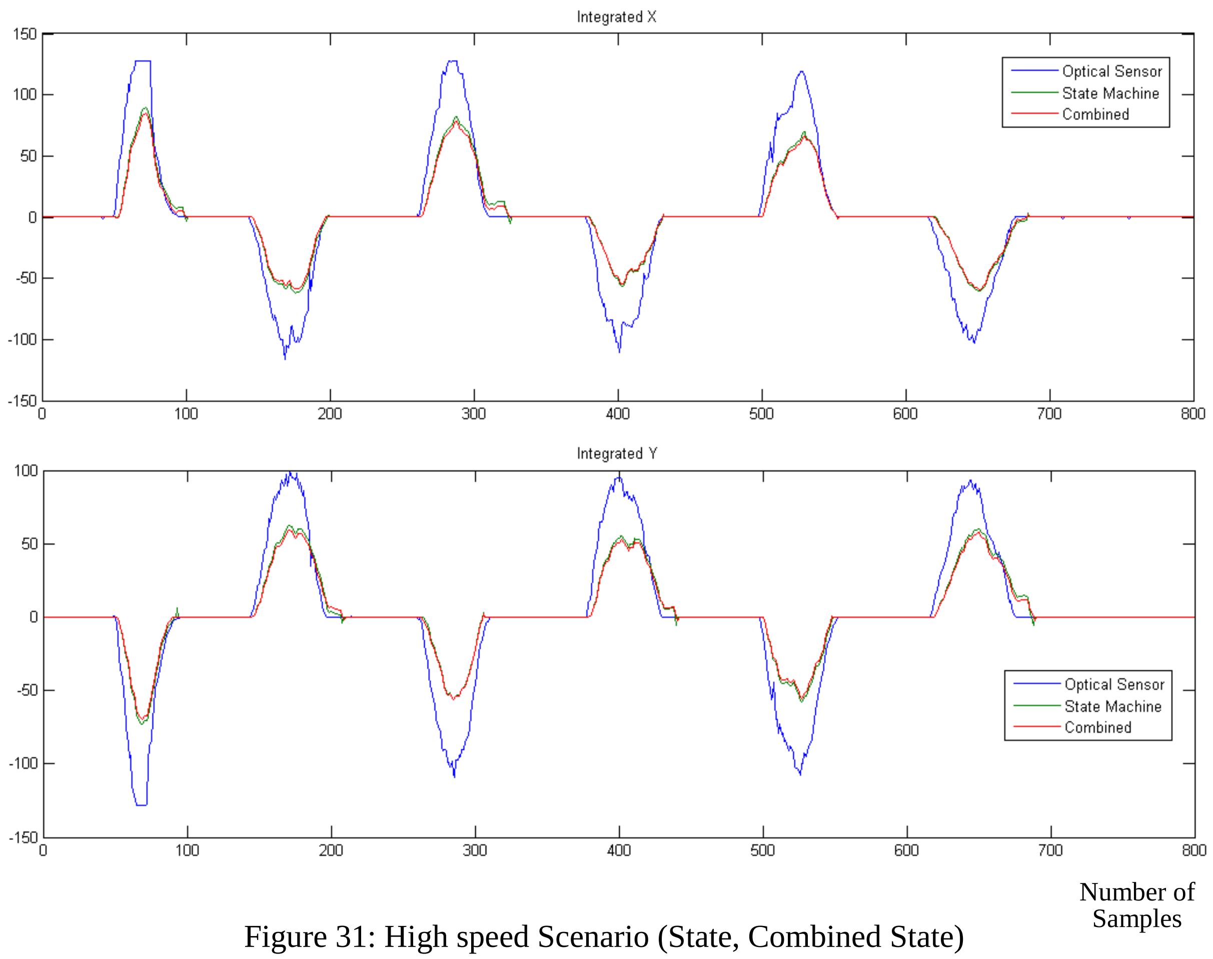


Figure 31: High speed Scenario (State, Combined State)

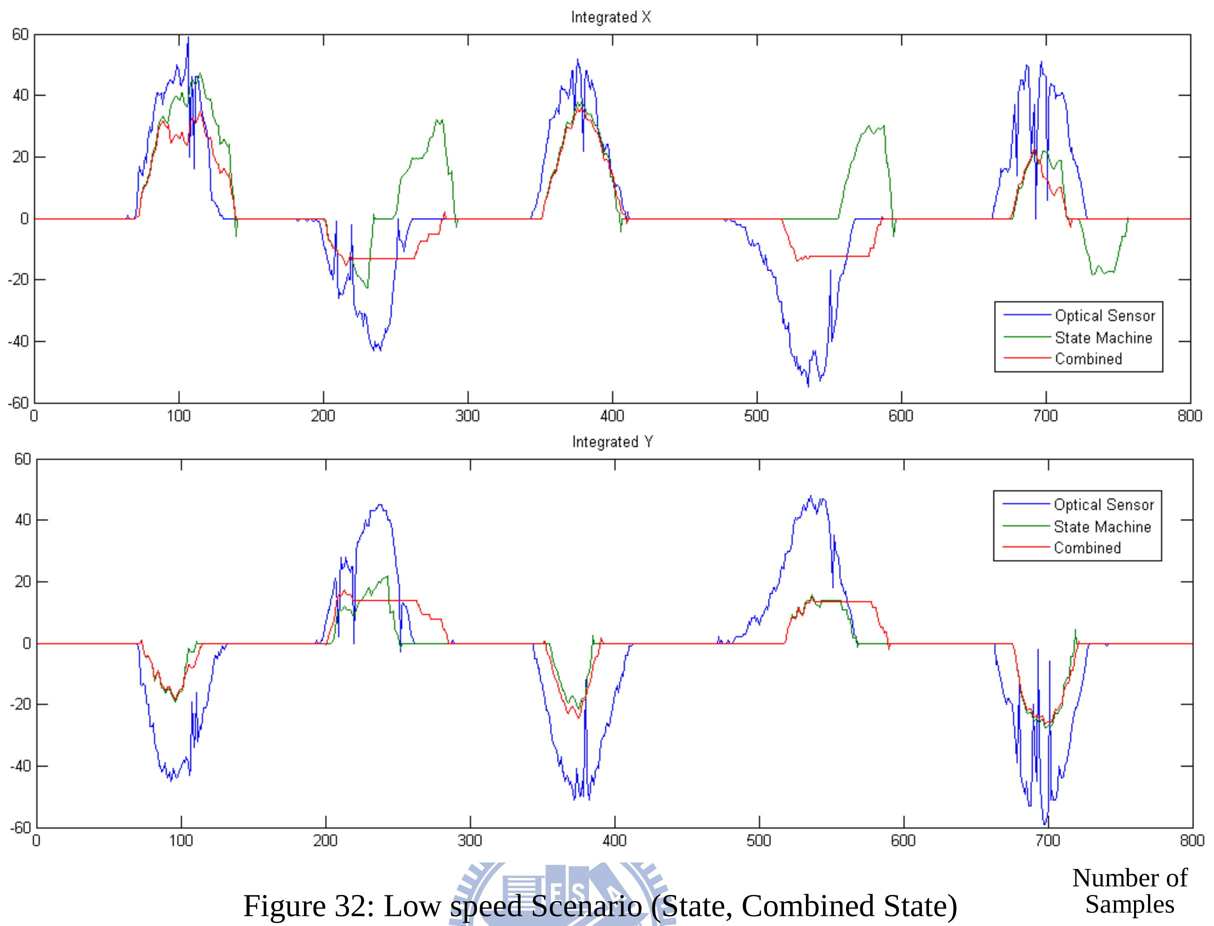


Figure 32: Low speed Scenario (State, Combined State)

For the high speed scenario (Figure 31), the curves are almost identical, because both are submitted to the same integration equations. However, in the low speed scenario (Figure 32), it is possible to notice that in some cases the combined state-machine avoids the negative residual velocity. The main reason for that is because when considering both axes to trigger a transition between two states, it would have less misinterpretation of the acceleration. For example, the transition between STABILIZING state to IDLE state. This transition only should occur if the prototype is not moving. If only one axis is analyzed, the probability to be stationary is smaller if both axes are analyzed.

Combined State-Machines Comparison

In this case, it is shown the advantages of adding a motion detection sensor

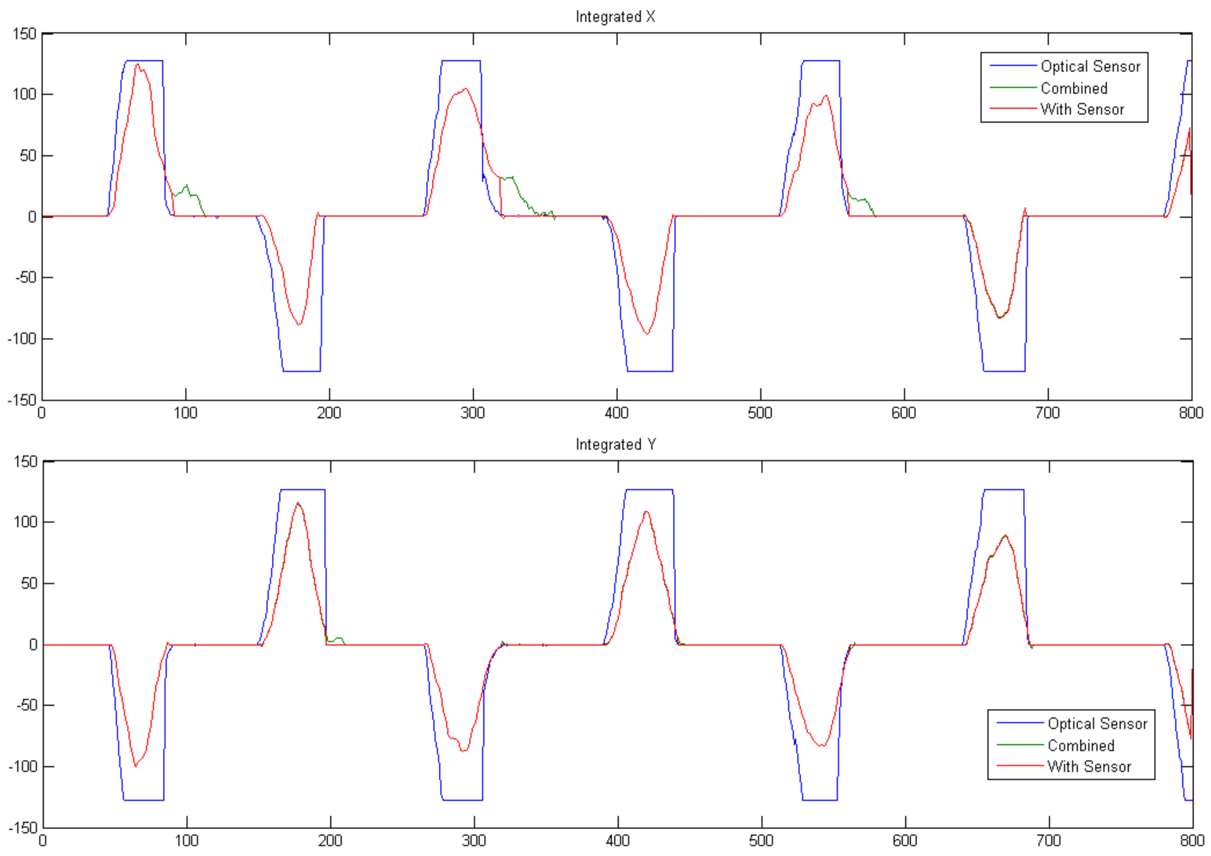


Figure 33: High speed Scenario (Combined State, Combined State with Motion Sensor)

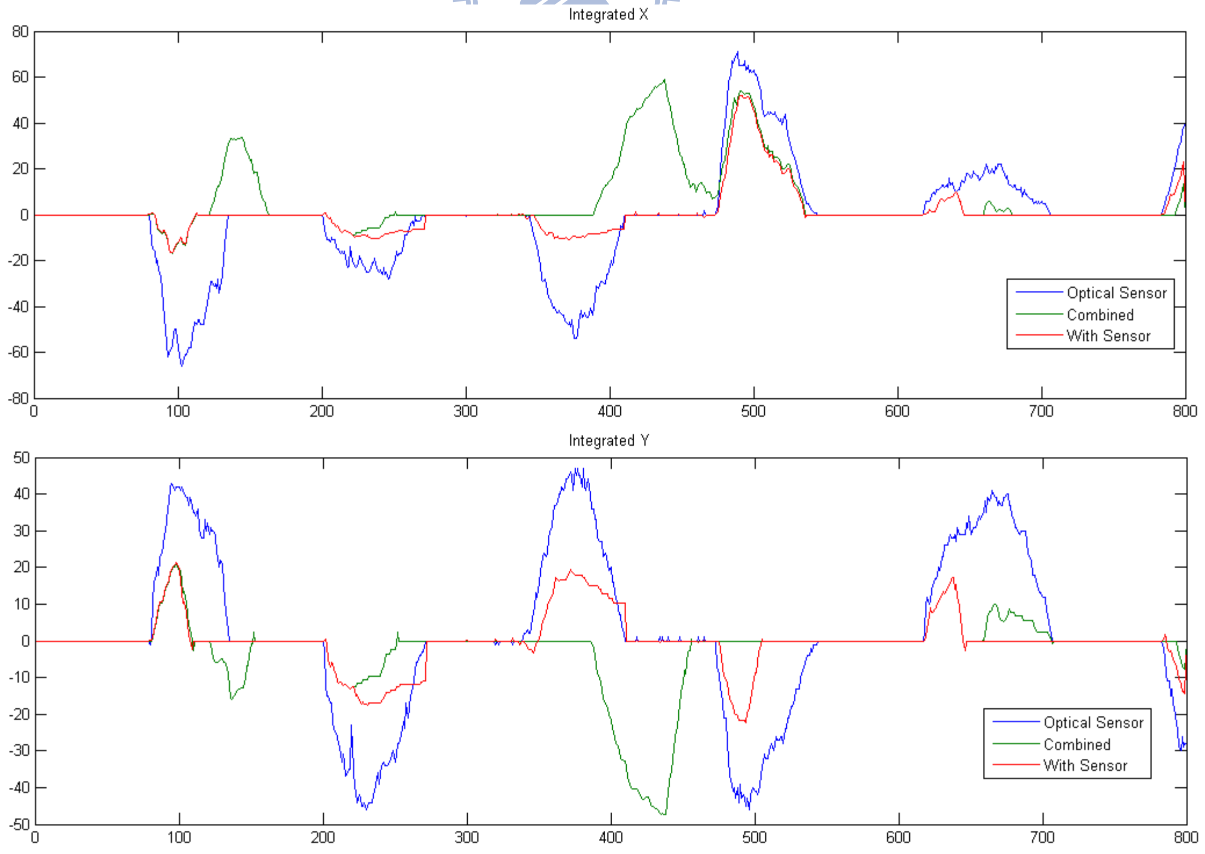


Figure 34: Low speed Scenario (Combined State, Combined State with Motion Sensor)

Both solutions use the same actions and states; the unique difference is the conditions to trigger the transitions between states. Adding a second sensor to detect the beginning and the ending of the movement can be really helpful. For the high speed scenario (Figure 33), the detection of any acceleration peak is not a problem. Therefore, it is possible to notice that both solutions behave exactly the same till the end of the movement. Using the extra sensor, it is possible to set the velocity immediately to zero, avoiding the positive residual velocity. For the low speed scenario (Figure 34), the extra sensor can be really useful to detect the exact instant that the acceleration should be integrated. Also, the decrement of the velocity during LOW SPEED state from the Combined state-machine with Motion sensor can be smaller to the solution without this extra sensor, since the end of the movement can be easily detected, which would avoid the positive residual velocity.

4.1.2 Mathematical Results

Using the same method to collect the data used to plot the graphics from the last section, a mathematical analysis is obtained, which considered four indicators:

- **Negative Integration Error:** it is measured in seconds and represents the interval of time starting at the moment that the estimated velocity crosses to zero and ending at the moment that the optical sensor velocity crosses to zero.
- **Positive Integration Error:** it is measured in seconds and represents the interval of time starting at the moment the optical sensor velocity crosses to zero and ending at the moment that the estimated velocity crosses to zero.
- **Reverse Velocity Area:** it is a percentage indicator obtained by dividing the total area of negative residual velocity with the total area of the velocity curve, which represents the total distance travelled.
- **Forward Velocity Area:** it is a percentage indicator obtained by dividing the total area of positive residual velocity with the total area of the velocity curve, which represents the total distance travelled.

The two scenarios are analyzed – low speed and high speed scenarios. The results are showed in the Table 7 and Table 8.

High Speed Scenario Total time: 22 s / 8.2 s Total Distance: 102.4 cm	Negative Integration Error	Positive Integration Error	Total Integration Error	Reverse Velocity Area	Forward Velocity Area	Total Error Area
Fuzzy Neural Network	1.936	1.111	3.047	2.629	2.0383	4.6673
Kalman Filter	1.078	2.376	3.454	1.5466	3.3538	4.9004
State Machine	0.011	1.837	1.848	0	3.82	3.82
State Machine with Kalman Filter	0.044	1.276	1.32	0	3.83	3.83
Combined State Machine	0.066	1.441	1.507	0	2.75	2.75
Combined State Machine with Sensor	0.066	0.055	0.121	0	0	0
18 movements	seconds	seconds	seconds	%	%	%

Table 7: Mathematical Analysis in High Speed Scenario

Low Speed Scenario Total time: 22 s / 10.3 s Total Distance: 39.9 cm	Negative Integration Error	Positive Integration Error	Total Integration Error	Reverse Velocity Area	Forward Velocity Area	Total Error Area
Fuzzy Neural Network	6.358	0	6.358	54.48	0	54.48
Kalman Filter	5.662	0	5.662	48.43	0	48.43
State Machine	0.314	2.753	3.067	12.12	2.823	14.943
State Machine with Kalman Filter	0.374	2.211	2.585	14.2	2.8365	17.0365
Combined State Machine	0.517	0.528	1.045	0.53	2.82	3.35
Combined State Machine with Sensor	0.455	0.066	0.521	0	0.0034	0.0034
18 movements	seconds	seconds	seconds	%	%	%

Table 8: Mathematical Analysis in Low Speed Scenario

In the tables, the total integration error is the sum of the positive and negative integration errors. The total error area is the sum of the reverse velocity area and the forward velocity area.

The first scenario indicates the total time of the experiment was 22 seconds and the prototype was only moving during 8.2 seconds. The total distance travelled was 102.4 centimeters. During the experiment, the mouse was moved 18 times.

The second scenario indicates the total time of the experiment was also 22 seconds, but during 10.3 seconds the prototype was moving. The total distance travelled was 39.9 centimeters and the mouse was moved 18 times in low speed.

It is possible to notice that the second scenario presents worse indicators, as observed in the graphics plotted in previous section.

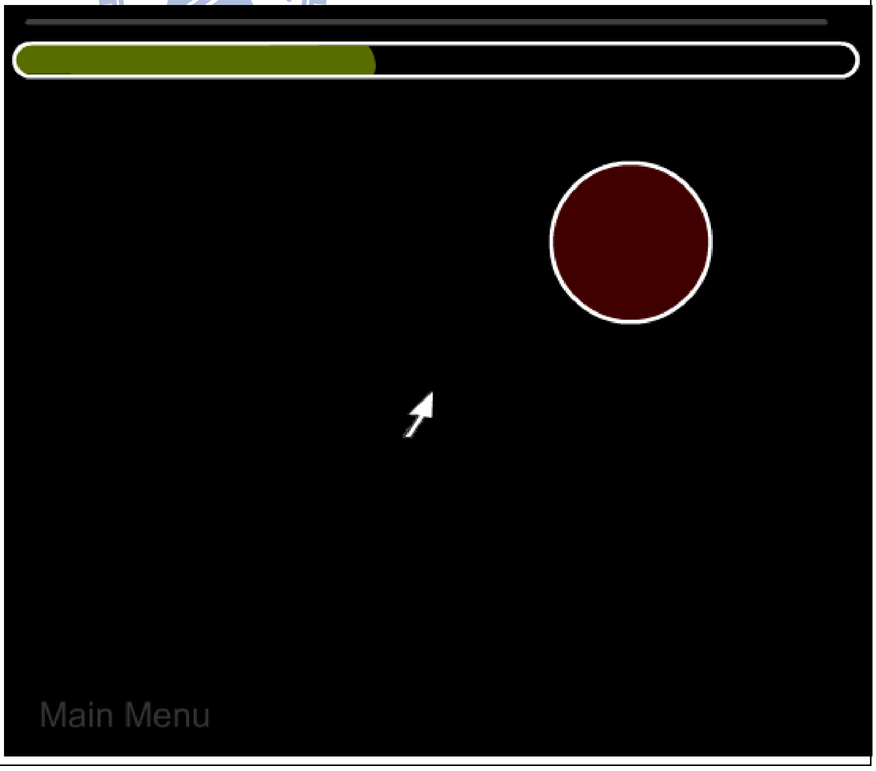
4.2 The Test Scenario

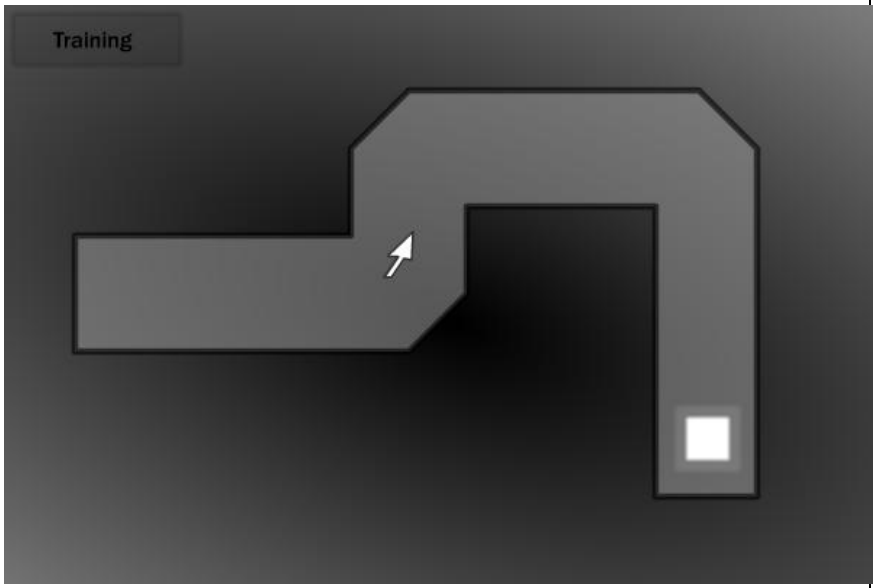
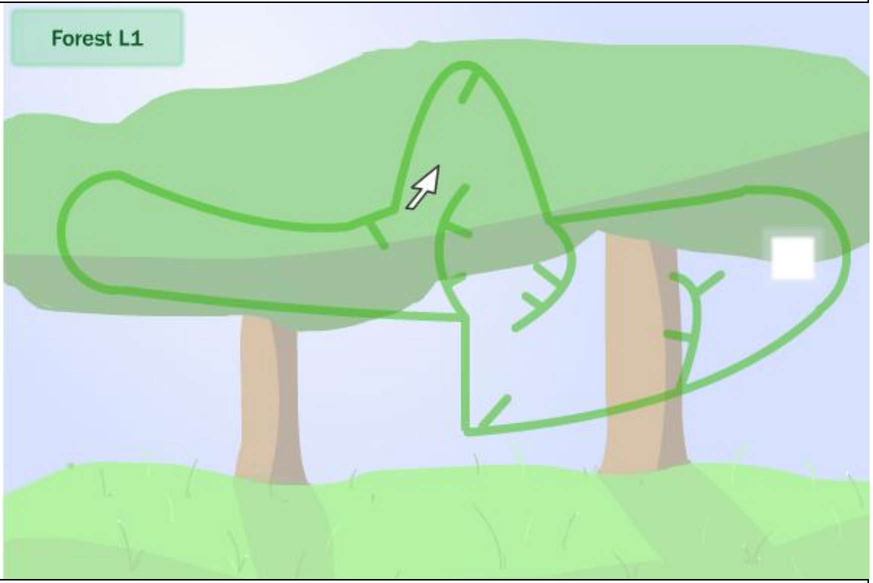
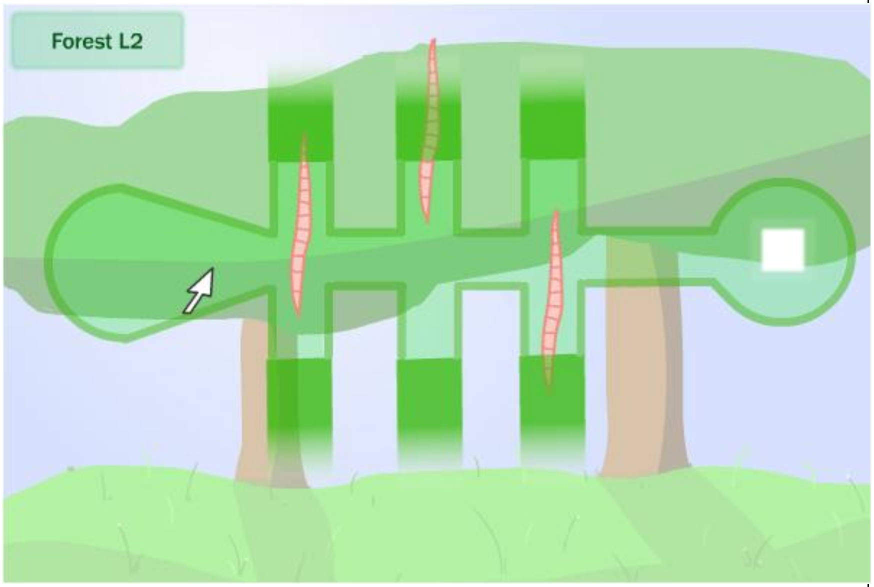
The test scenario proposed in this thesis consists in asking different users to use the prototype as a mouse device. Each user has to perform a specific number of tasks that use only the movement of the cursor on the screen. The test scenarios that each user has to run are based on FLASH applications that can easily be accessible in the internet. The applications used in this test environment can be executed in the following websites:

http://www.funny-games.biz/the-mouse-101.html

http://www.surfnetkids.com/games/mouse_1_0.htm

On Table 9, all test cases are described, including what the user should do and what this specific test is willing to measure.

Movement Response In this test the user should move the cursor to the red ball, which will disappear at the moment the cursor touches it and appear to another place. The application will count how many times the user can reach the ball in 30 seconds.	

Simple Path In this test the user should move the cursor until the white square without touching the walls. The idea is to check if the user can simply move the cursor using the prototype through a simple path.	
Complex Path In this test the user should perform the same task as the previous one; however the path is more complicated, including narrow passages.	
Synchronized Path In this test there are some moving obstacles that must be considered, therefore the user should move the cursor in synchronization with the obstacles. (*the worms will block the way periodically*)	

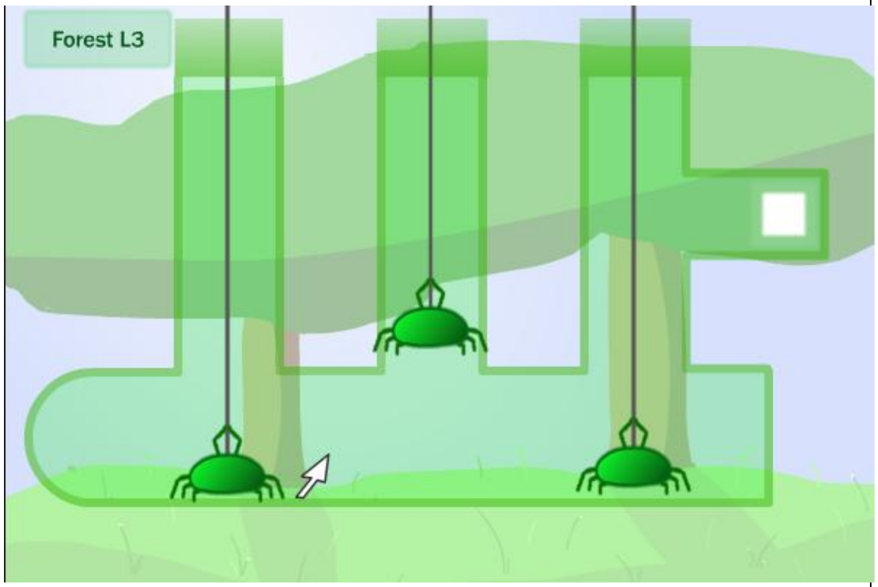
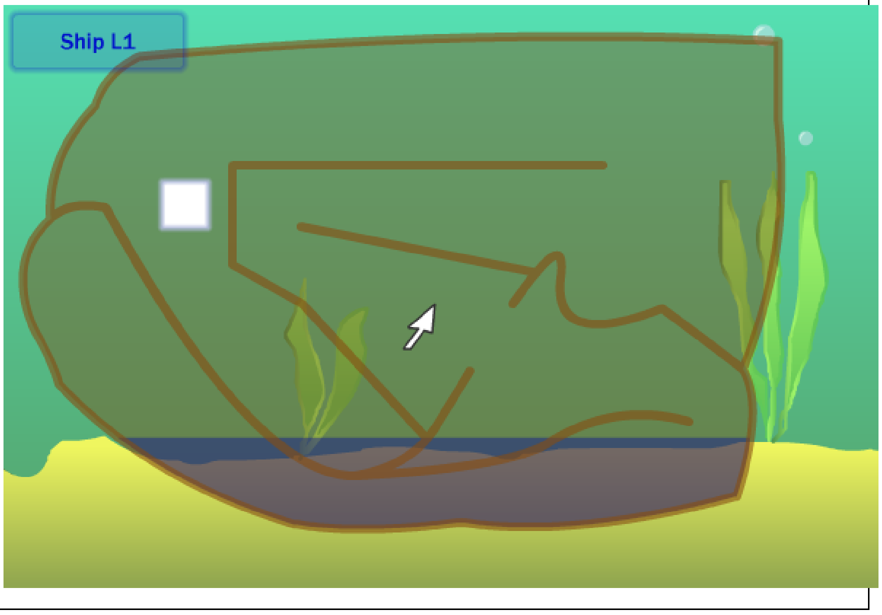
Synchronized Path with Time Constraint In this test besides to perform a synchronized movement the user should do it in a limited duration, since the last obstacle will block the passage for some time. (*the spiders will move up and down*)	
Long Movement with Time Constraint In this test the user should move the cursor for a long path in a short period of time, because after a while, the scenario will be inundate with water coming from the bottom.	

Table 9: Test cases for the performance test

4.3 Performance Results

The test bench results are scored in the following way:

- *The movement response test is scored by counting number of object hits;*
- *The other tests are scored based on the number of failures. If the user can pass the test in his first attempt, he gets 3 points. If he passes in his second chance, he*

gets 2 points. In his third chance, he gets 1 point. And if in three attempts, he could not pass any time, no points are scored.

Twelve people were invited to execute the tests, which means that the best score mouse device can have is 12 x 3 = 36 points. For the first test, the average of the performance of each user is calculated.

There are eight mouse devices to be tested, where seven of them use the prototype and only one use a regular commercial mouse device **[Normal Mouse]**. Only the best performance techniques obtained in the section 4.1 are ported to the microcontroller from the prototype and tested in the test environment described in the previous section, which includes:

- Combined XY Axes State-machine **[Combined State-Machine (SM)]** (section 3.4.3);
- Combined State-Machine with Optical Sensor as Motion Detection Sensor **[Combined SM with Optical Sensor]** (section 3.4.4).

When using the above techniques, the best sample rate obtained was around 91 samples per second, which corresponds to a period of 11 milliseconds. Therefore, one of the firmware to be tested uses the optical sensor that sends 91 packets per second to the host machine **[Prototype Mouse (11 ms)]**. Moreover, the packets are pipelined three times, adding a delay of 33 milliseconds. In this way, it is possible to verify if adding a delay and decreasing the sample rate will interfere in the performance. The case where no delay and pipeline are added also is tested, which will indicate a sample rate of 125 packets per second or period of 8 milliseconds between two packets. **[Prototype Mouse (8 ms)]**.

Other techniques ported to the prototype and tested are:

- Combined State-Machine with Physical Button **[Combined SM with Button]** (section 3.4.5);

- Combined State-Machine with Microphone as Motion Detection Sensor **[Combined SM with Microphone]** (section 3.4.4).
- State-Machine with Free-Movement **[Free-Movement]** (section 3.4.6).

On next, the test benches of each test are illustrated:

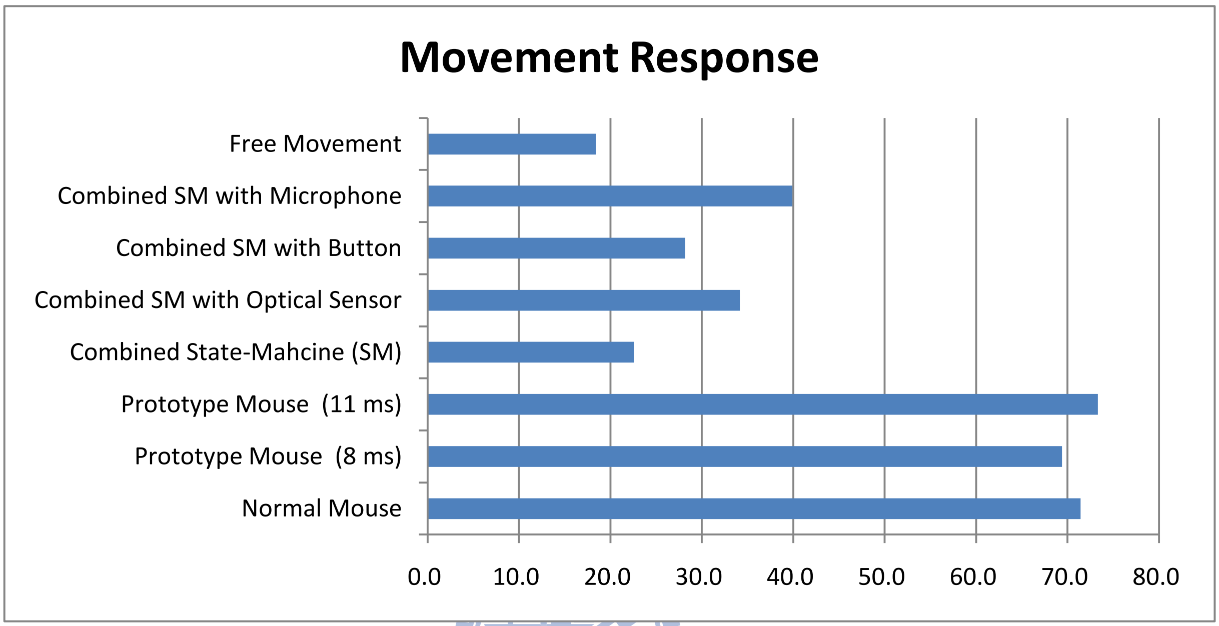


Figure 35: Movement Response test bench

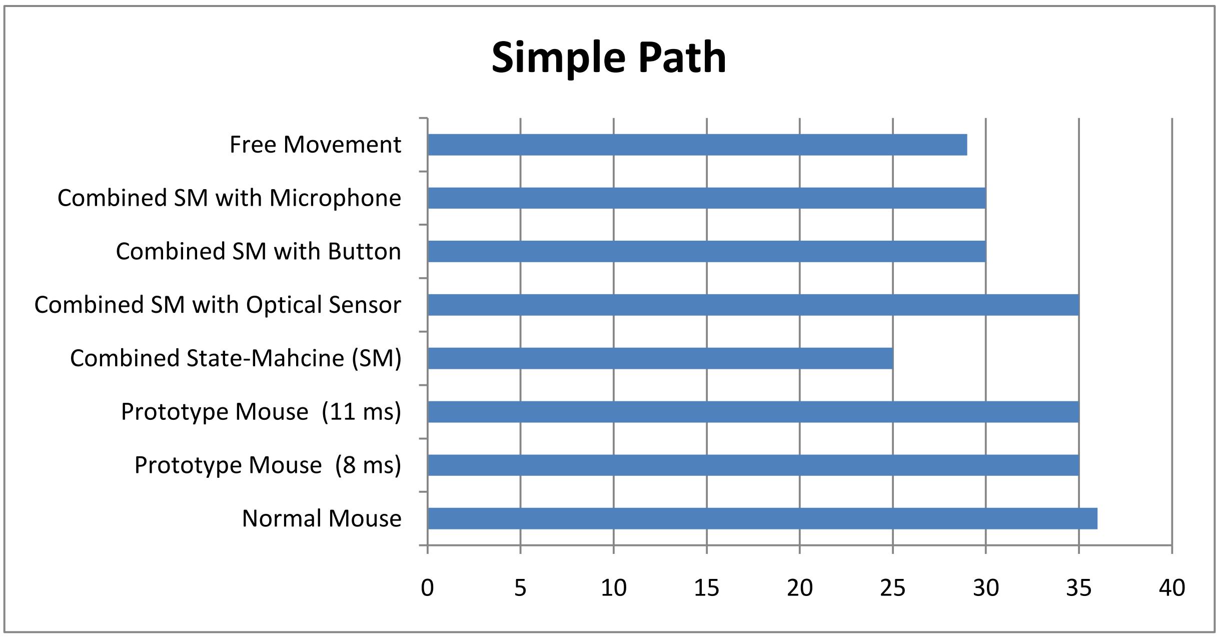


Figure 36: Simple Path test bench

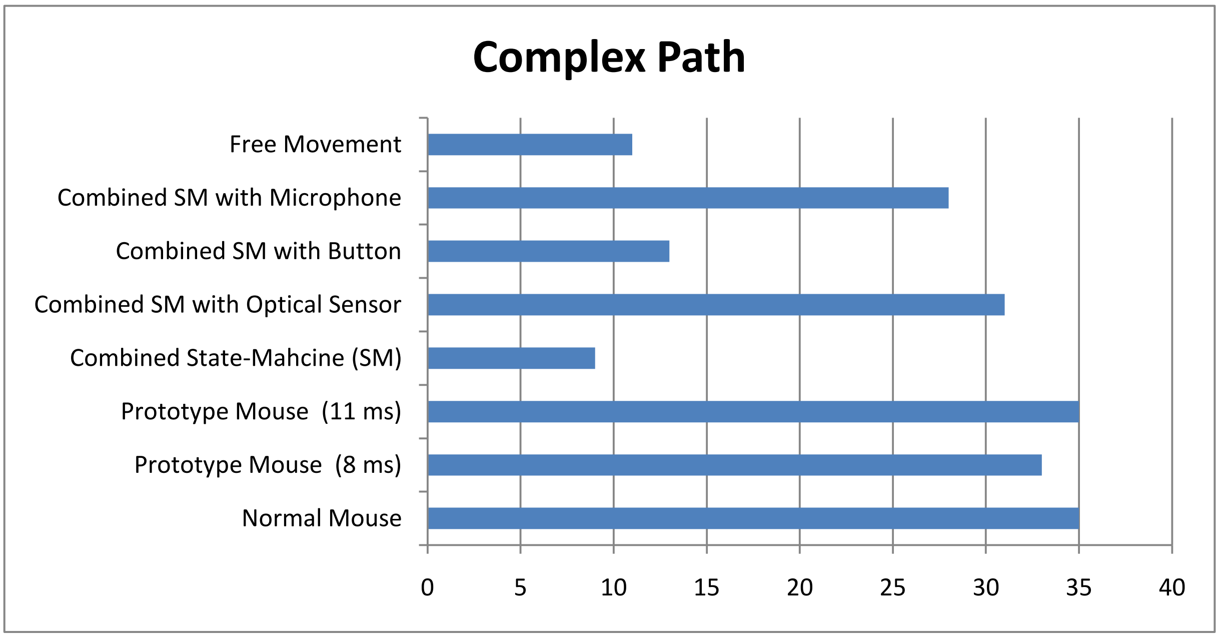


Figure 37: Complex Path test bench

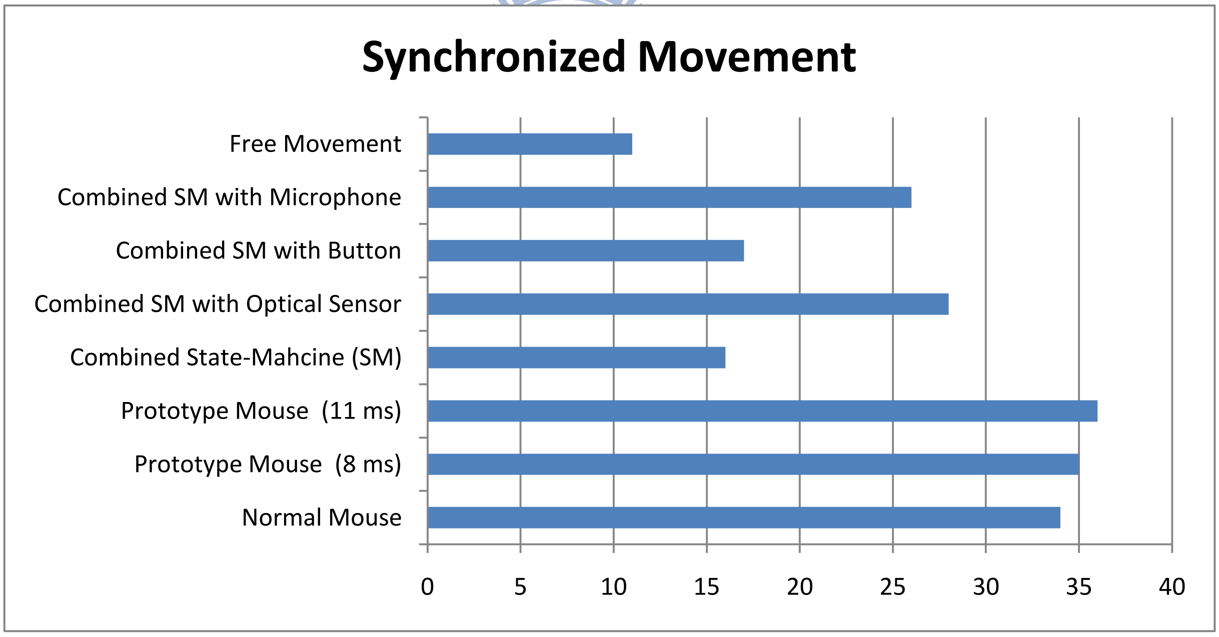


Figure 38: Synchronized Movement test bench

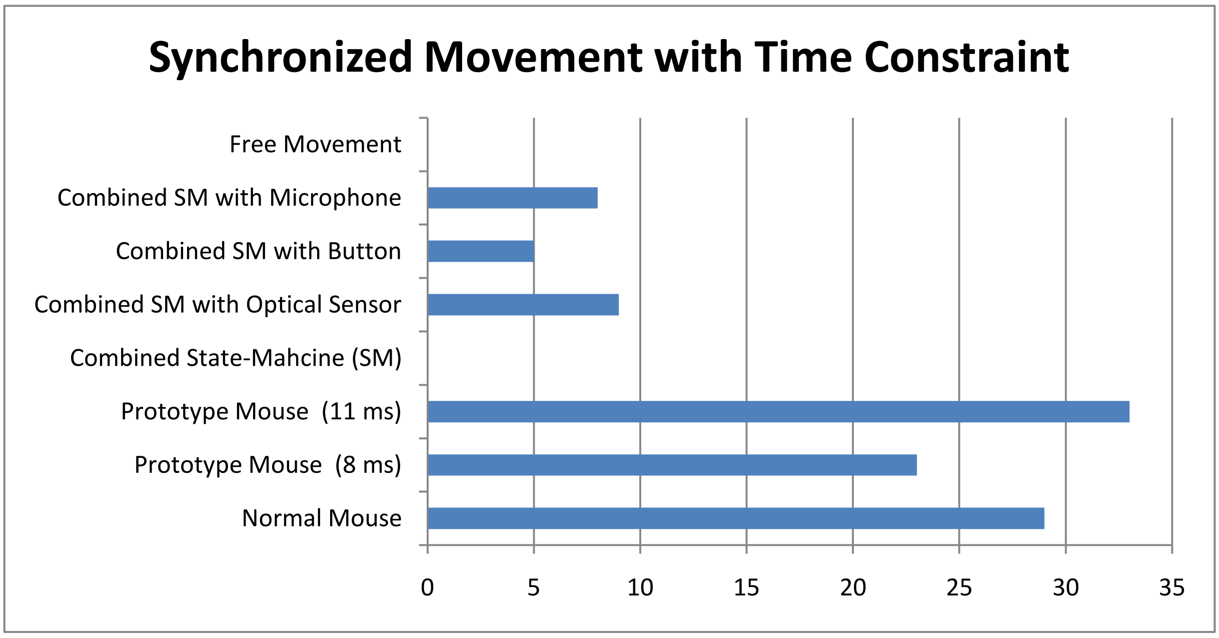


Figure 39: Synchronized Movement with Time Constraint test bench

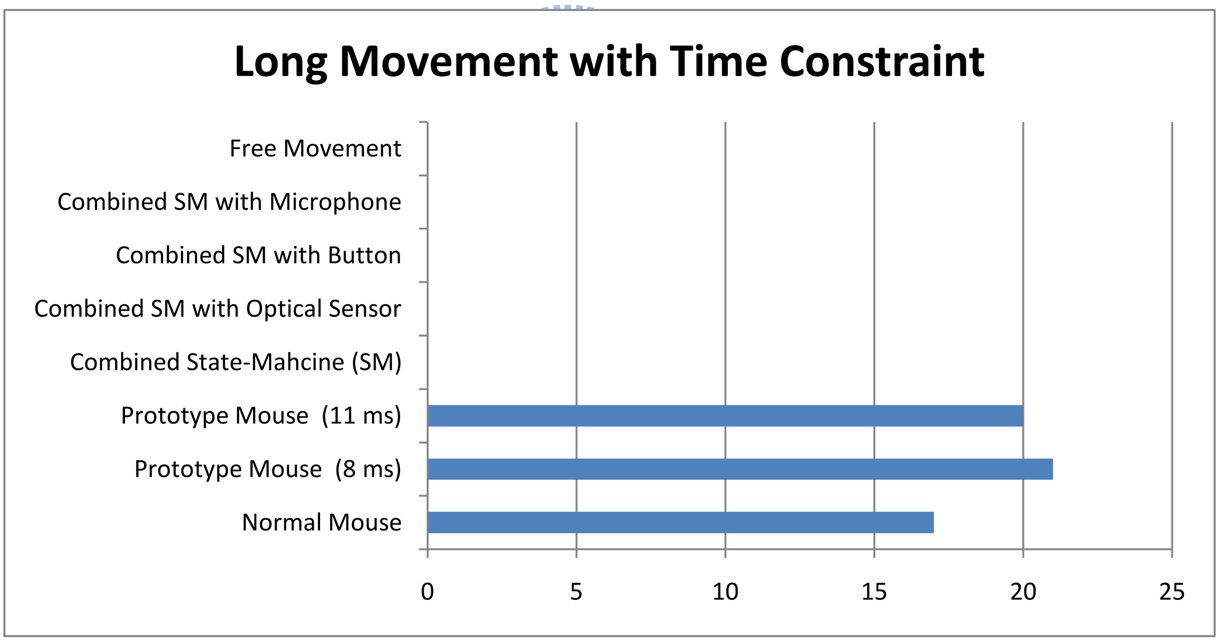


Figure 40: Long Movement with Time Constraint test bench

When analyzing only the mouse devices based on optical sensor, the test benches are quite similar, which indicates that there is almost no difference between a regular commercial mouse and the prototype based on optical sensor. Observing only the velocity of the cursor on screen, it was possible to notice that the “Prototype Mouse (11 ms)” runs faster than others, which make a better mouse to solve some specific tasks, especially the ones based with time constraint. Sometimes the “Normal Mouse” has worse performance the others because it was the first one to be tested, so after the user tested it and failed in the first attempts, he could improve his performance on the next tests.

Analyzing only the mouse devices based on accelerometer, it is possible to notice that the combination of the combined state-machine with motion detection sensor has the best performance, reaching scores almost as good as the optical sensors when there is no time constraint.

Chapter 5. Conclusions and Future Work

An inertial mouse device based on 3-axis accelerometer is proposed and evaluated in this thesis. The construction of a prototype containing an optical sensor and a 3-axis accelerometer allowed a very accurate comparison between different estimator techniques based on integration of acceleration and an equivalent commercial mouse device for two-dimensional use. Moreover, having a reference model based on the optical sensor, it is possible to tune parameters when estimating the velocity and/or train network structures.

The experimental results presented in chapter 4 show that formal mathematical models using Kalman Filter; or probabilistic techniques and pattern association using Fuzzy-Neural Network are not enough to have an accurate estimation of the movement. However, when breaking the movement in small parts and classify each part in different states and defining specific transitions between the states, better results can be obtained in some cases, specially for short movements in low and high speed. But for long movements with changes in the direction, as the sinusoidal case, this algorithm will simply ignore the movement.

Besides the graphical and mathematical analysis of each estimation technique, a test environment is proposed to simulate the use of a mouse device. From all techniques that use the accelerometers, the combined state-machine using motion detection sensor has the best performance. The digital microphone reached a performance as good as using optical sensor for motion detection. Both solutions can easily be used as a mouse device to perform the majority of tasks when interacting with the computer. The unique tasks that would not perform well are those that present time constraints.

As suggestion for future work, new motion detection sensors should be experimented to have a larger portfolio of options, and then compare their cost, size and power consumption, which are aligned with the advantages of the inertial mouse devices. Also, a new prototype should be designed using only inertial sensors and motion detection sensors; and a faster processor should be used to decrease the sampling rate, which can contribute significantly to the performance when using as a mouse device. The use of a FPGA is highly recommended, which will allow processing many sensors at the same time, besides that all states-machine solutions proposed in this thesis are easily implementable in hardware.

References

[1] L. Olson, “Inertial Mouse System”, US Patent (4,787,051), 1988.

[2] Grewal, M.S.; Henderson, V.D.; Miyasako, R.S.; , "Application of Kalman filtering to the calibration and alignment of inertial navigation systems," *Automatic Control, IEEE Transactions on* , vol.36, no.1, pp.3-13, Jan 1991.

[3] Helmi, N.; Helmi, M.; , "Applying a neuro-fuzzy classifier for gesture-based control using a single wrist-mounted accelerometer," *Computational Intelligence in Robotics and Automation (CIRA), 2009 IEEE International Symposium on* , vol., no., pp.216-221, 15-18 Dec. 2009.

[4] Liu, H.; Pang, G.; , "Accelerometer for mobile robot positioning," *Industry Applications Conference, 1999. Thirty-Fourth IAS Annual Meeting. Conference Record of the 1999 IEEE* , vol.3, no., pp.1735-1742 vol.3, 1999

[5] Lin, C.-T., & Lee, C. S. G. “Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*”. Upper Saddle River, NJ: Prentice Hall*, 1996.

[6] Seongbae Lee; Gi-Joon Nam; Junseok Chae; Hanseup Kim; Drake, A.J.; , "Two-dimensional position detection system with MEMS accelerometer for mouse applications," *Design Automation Conference, 2001. Proceedings* , vol., no., pp. 852- 857, 2001.

[7] Kurt. Seifert, and Oscar Camacho, “Implementing Positioning Algorithms using Accelerometers”, *Freescale Semiconductor Application Note*, 2007.

[8] Tracey, M.; Winters, J.; , "Neuro-fuzzy advisor for mouse setting in Microsoft Windows," *[Engineering in Medicine and Biology, 1999. 21st Annual Conf. and the 1999 Annual Fall Meeting of the Biomedical Engineering Soc.] BMES/EMBS Conference, 1999. Proceedings of the First Joint* , vol.1, no., pp.664 vol.1, 1999

[9] Catlin, Donald E. "Estimation, control, and the discrete kalman filter", *In Applied Mathematical Sciences 71*, 1989.

[10] G. Welch, and G. Bishop, "An Introduction to the Kalman Filter", *University of North Carolina at Chapel Hill, Department of Computer Science*, 2004.

[11] "MMA7360L Data Sheet", Freescale Semiconductor, http://www.freescale.com/files/sensors/doc/data_sheet/MMA7360L.pdf, 2007

[12] "PIC18F2458/2553/4458/4553 Data Sheet", Microchip, http://ww1.microchip.com/downloads/en/DeviceDoc/39887c.pdf, 2007

[13] "ADNB-6011-EV and ADNB-6012-EV Data Sheet", Avago Technologies, http://www.avagotech.com.tw/docs/AV02-1410EN, 2006.

[14] Hegner, H.; Skovsgaard, T.; , "Estimating acceptable noise-levels on gaze and mouse selection by zooming," *Student Paper, 2008 Annual IEEE Conference* , vol., no., pp.1-4, 15-26 Feb. 2008.

[15] "Mouse (computing)", Wikipedia, http://www.wikipeida.org/wiki/Mouse_(computing)

[16] M. S. Grewal and A. P. Andrews, *Kalman Filtering : Theory and Practice Using MATLAB*, 2nd ed. Wiley-Interscience, January 2001.

[17] Chin-Woo Tan; Sungsu Park; , "Design of accelerometer-based inertial navigation systems," *Instrumentation and Measurement, IEEE Transactions on* , vol.54, no.6, pp. 2520- 2530, Dec. 2005.

[18] Liu, R.; Ming Liu; Xiaokun Sun; Yawen Wei; , "Signal Processing and Accelerometer-based Design for Portable Small Displacement Measurement Device," *Embedded Software and Systems, 2008. ICESS '08. International Conference on* , vol., no., pp.575-579, 29-31 July 2008.

[19] Hsu-Yang Kung; Chin-Yu Ou; Shin-Di Li; Chun-Hao Lin; Hong-Jie Chen; Yu-Lun Hsu; Miao-Han Chang; Che-I Wu; , "Efficient movement detection for

human actions using triaxial accelerometer," *Consumer Electronics (ICCE), 2010 Digest of Technical Papers International Conference on* , vol., no., pp.113-114, 9-13 Jan. 2010.

[20] Ching-Chang Wong; Chi-Tai Cheng; Hao-Che Chen; Yue-Yang Hu; Chii-Sheng Yin; , "Static balancing control of humanoid robot based on accelerometer," *SICE Annual Conference, 2008* , vol., no., pp.2836-2840, 20-22 Aug. 2008.

[21] Piedrahita,; Andres, Giovanny; Guayacundo,; Marcela, Diana; , "Evaluation of Accelerometers as Inertial Navigation System for Mobile Robots," *Robotics Symposium, 2006. LARS '06. IEEE 3rd Latin American* , vol., no., pp.84-90, 26-27 Oct. 2006.

[22] North, E.; Georgy, J.; Tarbouchi, M.; Iqbal, U.; Noureldin, A.; , "Enhanced mobile robot outdoor localization using INS/GPS integration," *Computer Engineering & Systems, 2009. ICCES 2009. International Conference on* , vol., no., pp.127-132, 14-16 Dec. 2009.

[23] Blackmon, F.R.; Weeks, M.; , "Target acquisition by a hands-free wireless tilt mouse," *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on* , vol., no., pp.33-38, 11-14 Oct. 2009.

[24] Pandit, A.; Dand, D.; Mehta, S.; Sabesan, S.; Daftery, A.; , "A Simple Wearable Hand Gesture Recognition Device Using iMEMS," *Soft Computing and Pattern Recognition, 2009. SOCPAR '09. International Conference of* , vol., no., pp.592-597, 4-7 Dec. 2009.

[25] Dand, D.; Mehta, S.; Sabesan, S.; Daftery, A.; , "Handicap Assistance Device for Appliance Control Using User-Defined Gestures," *Machine Learning and Computing (ICMLC), 2010 Second International Conference on* , vol., no., pp.55-60, 9-11 Feb. 2010.

[26] Tiexiang Wen; Lei Wang; Jia Gu; Bangyu Huang; , "An acceleration-based

control framework for interactive gaming," *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE* , vol., no., pp.2388-2391, 3-6 Sept. 2009.