UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Hardware Strategies Applied to the Latency Reduction on Tactile Internet

José Cláudio Vieira e Silva Junior

Supervisor: Prof. Dr. Marcelo Augusto Costa Fernandes

Doctoral Thesis presented to the Graduate Program in Electrical and Computer Engineering of UFRN (concentration area: Computer Engineering) as part of the requirements for obtaining the title of Doctor of Science.

PPgEEC Order Number: D270
Natal, RN, February 2020

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Silva Junior, José Cláudio Vieira e.
Hardware strategies applied to the latency reduction on tactile internet / José Cláudio Vieira e Silva Junior. - 2020.
100 f.: il.

Tese (doutorado) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica e de Computação.
Orientador: Prof. Dr. Marcelo Augusto Costa Fernandes.

1. Tactile internet - Tese. 2. Latency reduction - Tese. 3. Haptic device - Tese. 4. Reconfigurable computing - Tese. 5. FPGA - Tese. I. Fernandes, Marcelo Augusto Costa. II. Título.

RN/UF/BCZM CDU 621.865.8

Elaborado por Ana Cristina Cavalcanti Tinoco - CRB-15/262

Acknowledgements

To my family, especially mother, father, and wife, for always supporting me in every way possible.

To Ana, João Batista, Brenda, and Caio, who gave me total support, welcoming me into their home during the whole journey.

To my supervisor, Dr. Marcelo A. C. Fernandes, for his teachings, encouragement, support and above all, his patience at all times.

To all colleagues at LAMII, who have always shown friendship and total willingness to help with whatever was needed.

To my doctoral colleagues, especially Sérgio Natan, Luís Feliphe, Lettiery, Leonardo and Virgínia, who have always helped and shared important moments of this work.

To Daniel Noronha, Matheus Torquato and Sérgio Natan, for their partnership in the tactile glove project, which resulted in the publication of a scientific article.

To the members of the examining board for their participation and valuable contributions.

To CAPES, for its financial support and opportunity to do an internship in the United Kingdom through the PDSE-CAPES program.

And, finally, to all those who somehow contributed to the achievement of this journey.

Resumo

Este trabalho se propõe a apresentar estratégias de hardware aplicadas a redução de latência na internet tátil. A motivação é estudar os desafios contidos no desenvolvimento do hardware associado aos dispositivos táteis, especialmente questões relacionadas ao limite de latência dos componentes do sistema. Como se sabe, para que um ambiente de internet tátil funcione de forma desejável, é necessário respeitar um limite mínimo de latência associada ao envio e a volta dos dados. Uma vez que algumas aplicações táteis permitem que alguns sentidos humanos possam interagir com as máquinas de forma remota, isso faz com que, quase sempre, o limite mínimo de latência associada ao envio e a volta dos dados fique com um atraso temporal na faixa dos milissegundos. Sendo assim, percebe-se que existe uma demanda por dispositivos táteis com elevado processamento. Diante deste contexto, são apresentadas três propostas de hardware que tem como objetivo principal reduzir a latência total produzida por este tipo de dispositivo. A primeira estratégia proposta para o desenvolvimento do hardware é usar computação reconfigurável (em FPGA) para minimizar o tempo de execução dos algoritmos associados ao dispositivo. A segunda proposta de hardware também faz o uso da computação reconfigurável (em FPGA), no entanto, o hardware é projetado usando outro tipo de representação numérica. Finalmente, a terceira proposta apresenta um modelo de luva tátil implementada usando um tipo de sistema microprocessado. Resultados associados as três propostas são apresentados e mostram a viabilidade das estratégias, apresentando um desempenho superior em relação aos trabalhos apresentados na literatura.

Palavras-chave: Internet Tátil, Redução de Latência, Dispositivo Háptico, Computação Reconfigurável, FPGA.

Abstract

This work proposes to present hardware strategies applied to reduce latency in the tactile internet. The motivation is to study the challenges contained in the development of the hardware associated with the tactile devices, especially issues related to the round trip latency limit of the system components. As is known, for a tactile internet environment to work desirably, it is necessary to respect a minimum limit of round trip latency. Since some tactile applications allow some human senses to interact with the machines remotely, this means that, almost always, the minimum limit of round trip latency has a time delay in the range of milliseconds. Thus, it is clear that there is a demand for tactile devices that are quite fast. In this context, three hardware proposals are presented that have the main objective to reduce the total latency produced by this type of device. The first strategy proposed for the development of hardware is to use reconfigurable computing (on FPGA) to minimize the execution time of the algorithms associated with the device. The second hardware proposal also makes use of reconfigurable computing (on FPGA). However, the hardware is designed using another type of numerical representation. Finally, the third proposal presents a tactile glove model implemented using a variety of micro processed system. Results associated with the three proposals are presented and show the viability of the strategies, presenting better performance concerning the works that were compared.

Keywords: Tactile Internet, Latency Reduction, Haptic Device, Reconfigurable Computing, FPGA.

Contents

List of Figures

List of Tables

List of Symbols

AI	Artificial Intelligence
CORDIC	COordinate Rotation DIgital Computer
CPD	Cartesian space Prediction and Detection
DMP	Digital Motion Processor
DoF	Degree of Freedom
ENV	Environment
ERM	Eccentric Rotating Mass
FBF	Feedback Force
FCS	Feedback Control System
FK	Forward Kinematics
FPGA	Field-Programmable Gate Array
FSM	Finite-State Machine
GPP	General-Purpose Processor
H2M	Human-to-Machine
HMD	Hardware associated with the Master Device
HSD	Hardware associated with the Slave Device
I2C	Inter-Integrated Circuit
IK	Inverse Kinematics
IMU	Inertial Measurement Unit
IoT	Internet of Things
JM	Jacobian Matrix
JPD	Joints space Prediction and Detection

KFF	Kinesthetic Feedback Force
LUT	Look Up Table
M2M	Machine-to-Machine
MD	Master Device
MSE	Mean Squared Error
MTS	Motion Tracking Sensor
NW	Network
OP	Operator
PC	Personal Computer
PO	PHANToM Omni
PWM	Pulse Width Modulation
RA	Reduction Rate of Area
SD	Slave Device
TCP	Transmission Control Protocol
TFB	Trigonometric Functions Block
UART	Universal Asynchrounous Receiver/Transmiter
VA	Vibration Actuator

Chapter 1

Introduction

Advances in research related to the networking area are enabling a new way of interacting with the internet. Called tactile internet, this new generation of internet will combine very low latency with high availability, reliability, and security (Dohler 2015). Another feature pointed out is that this new generation will be centered around applications that use human-machine communications (H2M) alongside devices that are compatible with tactile sensations (Aijaz et al. 2015, Berg et al. 2017). Initial studies on the tactile internet emerged in 2014 (Fettweis 2014, Union 2014), and from there, works on concepts, applications, and challenges were greatly explored (Moskvitch 2015, Dohler 2015, Maier et al. 2016, Simsek et al. 2016*a*).

Tactile internet will allow the human senses to interact with machines, involving not only audiovisual interaction, but also contact, integrating the human body with robotics and virtual reality systems (Maier et al. 2016). In a tactile internet environment, in the interaction cycle, the human operator exchanges information directly with a local device. This local device translates the information received from the operator and sends it over the internet through a bidirectional data communication network for a remote device to interact in the environment for the purpose of performing some type of task (Maier et al. 2016, Simsek et al. 2016*a*).

Figure 1.1 show a basic tactile system, in this kind of environment, haptic or tactile devices are often used as local devices, as they can transmit to the operator some kind of actions that represent physical characteristics related to the environment in which they are interacting. For the remote device, robotic manipulators are generally used. This kind of device has sensors capable of capturing characteristics of the environment, either real or virtual. These characteristics of the environment are sent as feedback from the remote device to the local device, which through the haptic interface transmits the feedback (physical stimuli) to the operator. This feedback information is intended to expand and refine the representation of real objects, or objects inserted into a virtual environment (Aijaz et al. 2015, Berg et al. 2017).

Naturally, in the infrastructure of the tactile internet environment, in addition to the communication network and local and remote devices, hardware equipment is also required. This hardware, in addition to interfacing with the communication network, is responsible for executing the algorithms associated with the devices. These algorithms usually involve the use of arithmetic operations and calculations of linear and nonlinear equations that need to be computed at high sampling rates to maintain application fidelity. All these components related to tactile internet infrastructure, individually generate a time delay and depending on the kind of application, the human reaction time can determine the desired time delay for communication between devices. This time delay defines the round-trip latency constraints that surround the technical specifications of the equipment

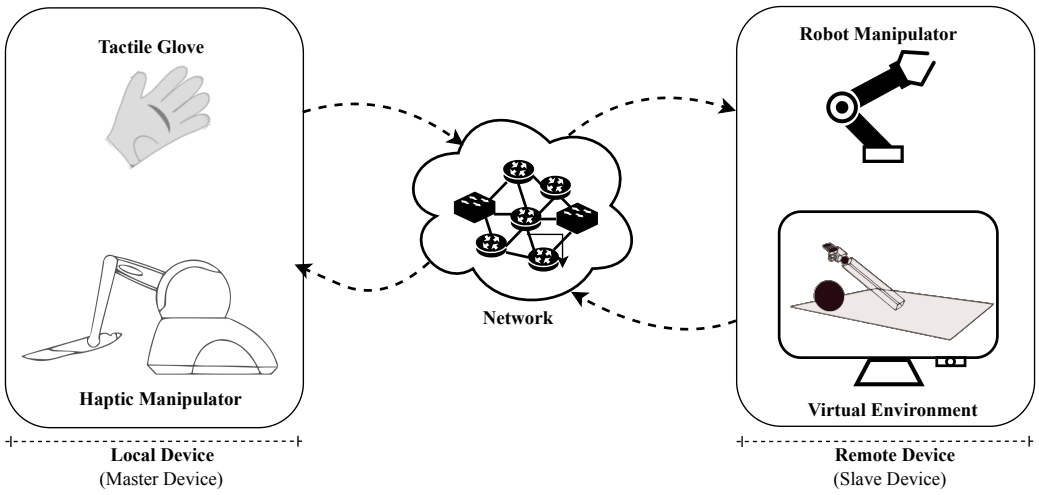


Figure 1.1: Basic tactile internet system.

that is part of the infrastructure of the tactile internet environment.

When the round-trip latency is above the desired time delay, the system may not respond properly, and as a result, the operator may suffer cybersickness, giving a sense of disorientation and discomfort (Fettweis 2014, Union 2014). Technically, this happens when the operator reacts to the feedback stimulus, at the wrong time and with an intensity that deviates from the correct values. Therefore, for tactile internet applications to work as desired (realtime), round-trip latency needs to be below the constraint limits. So, the lower the latency involved in the environment, the closer to the real will be the feel interaction with the operator. To shorten the response time involved in tactile internet infrastructure, strategies aimed at improving communications network latency and strategies aimed at improving hardware performance are being explored.

Some studies in the context of tactile internet that are related to communication networks can be easily found. The works (Aijaz 2016, Aijaz et al. 2015, Simsek et al. 2016*c*, Szabo, Gulyas, Fitzek, Fitzek & Lucani 2015), shows some types of techniques that can be used to improve network latency. However, this thesis focuses specifically on the hardware that is associated with the devices of the tactile internet environment. Therefore, several strategies can be used to improve the performance of hardware associated with devices. One factor that has a significant impact on hardware performance is the type of architecture it uses.

Microprocessor systems are widely used today due to the ease of embed the software that implements the algorithms. With this kind of architecture, software development can be simplified through the use of libraries just as the data transfer protocol can be fully defined via software. This hardware is designed for general use and applications are managed by an operating system that determines the order of execution. There is a demand for prototyping using this type of architecture in the context of tactile internet, but some works (Lin et al. 2018, Lobo & Trindade 2013, Arjun et al. 2018, Weber et al. 2016, Muramatsu et al. 2012), cannot reach the minimum latency limits. But in (Junior et al.

2019), the authors showed that it is possible to create a tactile application that respects latency limits using microprocessor systems.

The use of reconfigurable computing with Field-Programmable Gate Arrays (FPGAs) allows creating customizable hardware prototypes. They can be programmed to perform any digital function as they are made up of a large number of programmable units that can be configured to simulate the behavior of any circuit. In this kind of architecture, algorithms can be parallelized and optimized at the logical port level to speed up their operations. Works that use FPGAs integrated with haptic systems have been explored (Sánchez et al. 2010, Wu et al. 2014, Wong & Liu 2013, Linh et al. 2015, Jiang et al. 2017). Following the same way, it is also possible to find works that make use of hybrid architecture that combines reconfigurable computing in FPGA with microprocessor systems such as the example of the work (Gac et al. 2012).

It is notorious that these different types of hardware can be associated with devices and consequently used in the infrastructure of a tactile internet environment. Thus, this thesis proposes to make an analysis of the impact wich the different kinds of hardware architecture can influence on the reduction of latency in tactile internet applications. Regarding the use of reconfigurable hardware in FPGA, was created a generic tactile internet model that provides a haptic system that uses two haptic devices. The hardware associated with the device control algorithms were designed and synthesized in FPGA using two types of numeric representation, 32-bit floating-point and fixed-point using different format types. Besides performing an analysis of the different types of FPGA implementation of the control hardware, comparisons of these implementations were made with other works in the literature.

Another analysis was also needed to see the impact that hardware using microprocessor systems has on a tactile internet environment. For this, a tactile system that has two main devices were designed and created. The first is a glove-type tactile device that is used as a local device, and the second device, which is the remote device, used a virtual hand model that is modeled in a virtual environment. The control algorithms associated with the tactile glove were embedded in a hardware processor system on a chip (SoC). The virtual environment that provides virtual interaction was developed in a 3D virtual engine. In addition, analyses of the round-trip latency were performed and compared with other embedded systems applied to the tactile systems. This study has resulted on the first project called "Tactile glove device" that won first place in the local competition of the Intel Embedded Systems 2016 contest held at Instituto Metrópole Digital (IMD) in Natal, where was classified for the national stage of the competition, held at the VI Brazilian Symposium on Computer Systems Engineering (SBESC-2016) in João Pessoa. In the national stage of the project, it was in third place among approximately 60 projects from all over Brazil.

This thesis is inserted in a bigger context, which involves a research partnership with the research group commanded by Prof. Dr. Mischa Dholer at King's College London (KCL), England. The research group is the pioneer in research in the area of tactile internet, having extensive experience arising mainly from participation in major projects in the European community. Through the research internship funded by the Coordination for the Improvement of Higher Education Personnel (CAPES) through the Doctoral

Sandwich Abroad Program (PDSE), it was possible to develop part of this thesis at the KCL facilities. As a result, one article has been published and two are in the process of being published.

1.1 Objectives

The objective of this thesis is to contribute to the area of tactile internet, proposing hardware strategies in order to reduce the latency associated with the devices.

The specific objectives of this work are:

- Propose a specification of a new generic model of a tactile internet system;
- Develop a reconfigurable hardware reference model using parallelism techniques in FPGA of the modules of hardware that implements control algorithms associated with haptics devices;
- Develop a reference model of a tactile glove device implemented in a microprocessor system that delivers tactile feedback through the interaction with objects inserted in a virtual environment;
- Demonstrate through experiments, the viability of the strategies used integrating the hardware modules developed with the proposed tactile system.

1.2 Submitted and Published Articles

- JUNIOR, J.C.V.S.; Torquato, M.F.; Mahmoodi, T.; Dohler, M.; Fernandes, M.A.C. Reconfigurable Computing Applied to Latency Reduction on Tactile Internet. Submitted to IEEE Access.
- JUNIOR, J.C.V.S.; Torquato, M.F.; Fernandes, M.A.C. Parallel Fixed-Point Implementation of the Bilateral Control Algorithm Towards Haptic Device on FPGA. Submitted to IEEE Transactions on Circuits and Systems.
- JUNIOR, J.C.V.S.; Torquato, M.F.; Noronha, D.H.; Silva, S.N.; Fernandes, M.A.C. Proposal of the Tactile Glove Device. In MDPI Sensors. 2019, November, 19, 5029. DOI 10.3390/s19225029.

1.3 Thesis Outline

This thesis is organized in 4 chapters, as presented in the following paragraphs.

In this first chapter, an introduction was presented, in which the motivation and theme of the work are contextualized, in addition to the main objectives of this research and the published articles.

Chapter 2 presents a generic model of tactile internet as well as a detailed description of the development and implementations in FPGA of algorithms associated with haptic devices. In addition, to the results of the validation of the proposed hardware modules, the post-synthesis results of each implementation will be shown. Comparisons of the proposals implemented with the state of the artworks will also be presented. This Chapter is

based on the contents of the article *Reconfigurable Computing Applied to Latency Reduction on Tactile Internet* and contents of the article *Parallel Fixed-Point Implementation of the Bilateral Control Algorithm Towards Haptic Device on FPGA*

In Chapter 3, some related and state of the artworks will be presented, which relate to implementations of a tactile glove device. Details of hardware and software projects of a tactile glove, as well as the virtual environment, are described. Results and analyses on the round-trip latency time in the tactile environment of the Internet are developed and compared with other works in the literature. This Chapter is based on the contents of the article *Proposal of the Tactile Glove Device*.

Finally, in Chapter 4 the final considerations are presented, showing the conclusions about the results obtained and the possibility of future work.

Chapter 2

Reconfigurable Computing for Haptic Control

This chapter aims to present, analyze and evaluate a hardware reference model that uses reconfigurable computation (FPGA) for modules that implement nonlinear positioning and force calculations as well as a tactile system formed by two robotic manipulators. In addition to presenting the implementation details, simulations and experimental tests are performed in order to validate the proposed model. Results associated with the sampling rate, throughput, latency and post-synthesis occupancy area (in FPGA) are analyzed. The results are compared with other works in the literature with the objective of validating the performance of the proposed hardware, which stands out as having the overall best performance against the ones it was compared to.

2.1 Introduction

A tactile internet environment is basically composed of a local device (known as a master) and a remote device (known as a slave), where the master device is responsible for controlling the slave device over the internet through a two-way data communication network (Maier et al. 2016) (Simsek et al. 2016*a*). Bidirectional communication is needed to simulate the physical laws of action and reaction, where action can be represented as sending operational commands and reaction can be represented as the forces resulting from that action. In tactile internet applications, the desired time delay for device communication is characterized by an ultra-low latency. In bilateral communication, the required latency ranges from 1 ms (round trip) up to 10 ms depending on the application requirements (Li et al. 2018, Antonakoglou et al. 2018*a*, Nasrallah et al. 2018, Simsek et al. 2016*c*, Junior et al. 2019).

According to (Szabo, Gulyas, Fitzek, Fitzek & Lucani 2015), it can be noticed that in a tactile internet application, 30% of the total system latency is generated by the master and slave devices. These devices demand high processing speeds as repeated execution of a variety of high computationally expensive algorithms and techniques. These algorithms involve the use of arithmetic operations and calculations of linear and nonlinear equations that need to be computed at high sampling rates in order to maintain application fidelity. The remaining 70% of the latency is introduced by the communication network. However, current networks are not suitable for such latency constraints (Dohler et al. 2017). To minimize this problem, some research groups have been studying prediction techniques, where many algorithms have been studied and proposals using artificial intelligence (AI) have proved to be effective (Yu et al. 2015). On the other hand, the implementation of

complex AI-based prediction methods can further increase the latency of the computer systems present in master and slave devices, because inference and training require high computation resources (de Souza & Fernandes 2014).

Alternatively, new approaches such as reconfigurable computing can improve the performance of master and slave devices in a tactile system environment. Reconfigurable computing with field-programmable gate arrays (FPGAs) enables the creation of customizable hardware which allow algorithms to be parallelized and optimized at the logical gate level to speed up their operations. Literature results show that computationally expensive algorithms can achieve speedups of up to 1000× over software implementations when custom implemented in FPGAs (de Souza & Fernandes 2014, Da Costa et al. 2019, Coutinho et al. 2019, Torquato & Fernandes 2019, Da Silva et al. 2019, Lopes et al. 2019, Noronha et al. 2019).

In this context, this work proposes an implementation to target reducing the 30% of the total latency related to tactile devices. The project uses reconfigurable computation in FPGA to minimize the execution time of algorithms associated with master and slave devices. The use of reconfigurable computing allows the parallelization of algorithms and latency reduction compared to software systems embedded in traditional architectures with general purpose processors and microcontrollers. In an effort to validate the proposed strategy, this work presents a discrete reference model that can be adjusted for different types of master and slave devices in a tactile internet system. Validation results, throughput, and post-synthesis figures obtained for the proposed hardware implementation using FPGA reconfigurable computing are presented. Comparisons with other works in the literature show that the use of reconfiguration computing can significantly accelerate the processing speed in tactile devices.

2.2 Related Work

(N et al. 2018) presented a tactile internet environment that used a glove type device in conjunction with a robotic manipulator. The environment was developed using a general purpose processor, which made the execution of the algorithms sequential. In order to send the data, the tactile glove produced a latency of approximately 4.82 ms, and the hardware responsible for performing the inverse kinematics calculations took an interval of 0.95 ms. The latency values obtained in this application could be improved by hardware structures that allow algorithms parallelization.

Studies in the literature demonstrate the benefit of using FPGA to accelerate the sample rate for data acquisition from devices associated with haptic systems. Work (O'Malley et al. 2009) presented an implementation for controlling a 3-DoF (Degree of Freedom) device. The presented technique proposed to increase the device sampling rate using FPGA hardware together with a real-time operating system (RTOS) in order to increase the resolution acquisition of the stiffness sensor. The control technique presented was developed in 32-bit fixed-point and trigonometric functions were implemented using lookup tables.

The work described in (Tanaka et al. 2009) presented a control system for one-dimensional haptic devices (1-DoF). The FPGA control implementation used single-precision floating-point representation (IEEE std 754) and the algorithms performed all calculations in 50μs.

The processing time was satisfactory, however the data frame size to be sent over the network increased with the size of the DoF. This peculiarity can increase latency for more complex haptics systems with many DoFs. In the same topic of previous works, an implementation for bilateral control of single-dimensional haptic devices (1-DoF) was presented in (Franc & Hace 2013). A more accurate control techniques based on the sliding mode control (SMC) was implemented in FPGA, and to assist in performing the complex calculations, the CORDIC (COordinate Rotation DIgital Computer) was used. The hardware was designed to locally control two devices, one master and one slave. In the implementation, a 24-bit fixed-point was used, of which 9 bits in the integer part and 14 bits for the fractional, and the total execution time of the controllers was of 7.2375μs.

The works (O'Malley et al. 2009), (Tanaka et al. 2009) and (Franc & Hace 2013) presented a control that depends directly on the encoder reading of the device motors. Usually in commercial models, accessing the device electronics can be tricky requiring some reverse engineering and specific knowledge to make the appropriate encoder connections. On the other hand, some works abstract the data acquisition and work directly with robotics algorithms. These algorithms may require high computational power that can surpass the capabilities of many general-purpose processors (GPPs) that perform the operations sequentially.

Some studies demonstrate the benefit of using FPGA to accelerate robotic manipulation algorithms related to haptic systems. A hardware architecture implemented in FPGA for performing the forward kinematics of 5-DoF robots using floating-point arithmetic was described in (Sánchez et al. 2010). In this hardware implementation all the forward kinematics calculations were performed within $1.24\mu s$ which represents 67 clock cycles in a frequency of 54 MHz. The equivalent software implementation has a total processing time of 1.61036 ms. Overall, the hardware implementation is $1298\times$ faster than the software implementation, which means a considerable acceleration in the forward kinematics processing time.

The authors of the paper (Gac et al. 2012) presented an FPGA implementation of inverse kinematics, velocity calculation and acceleration of a 3-DoF robot. Three systems were created: the first one did not use any arithmetic co-processor and floating-point operations were performed in software; in the second system a floating-point co-processor was used which allowed the execution of the four basic mathematical operations in hardware; lastly, the third system also had a custom arithmetic co-processor but in this case it allowed hardware computation of square root. The overall times to perform the calculations were 2324μs, 560μs and 143μs and the total logic elements used from the entire device were 4501 (4%), 5840 (5%) and 7219 (6%), respectively. The work uses hardware-software to implement inverse kinematics, in which critical parts were implemented in FPGAs to accelerate the whole process.

In (Wu et al. 2014) is presented a hardware to control a 6-DoF device using 32-bit fixed-point representation, where 21 bits were used for the fractional part and 11 bits for the integer part. In that work, a CORDIC implementation was used to assist in performing the trigonometric calculations. The total time spent to compute the forward kinematics was 3μs and for the inverse kinematics the time was 4.5μs for a clock of 50 MHz. However, in the presented proposal, some calculations were performed sequentially, that is,

for the execution of the forward kinematics it was necessary 150 clock cycles and for the inverse, 225 cycles. The use of partial parallelization in the execution of robotic manipulation algorithms provided a significant increase in system throughput. Nevertheless, it is important to note that there is still room for improvement since all calculations can be computed in parallel.

Another hardware implementation of inverse kinematics was presented in (Wong & Liu 2013). The device used was a 10-DoF biped robot. A CORDIC implementation was used to perform the trigonometric calculations. The execution time needed to compute the kinematics of the 10 joints in FPGA was of 0.44μs. In this work, a comparison with a software implementation was also performed, and the time taken to perform the same calculations was 3342μs, i.e. the gain on execution, or speedup, on custom FPGA hardware was 7595$\times$. The resulting error between both implementations was acceptable for this specific control.

In (Linh et al. 2015) it was presented an FPGA implementation of the forward and inverse kinematics of a 5-DoF device. The hardware was developed using a fixed-point representation where 32 bits were used for the angles representation and 15 bits for the fractional part. For the device spatial positioning, 16 bits were used of which 7 bits for the fractional part. In the implementation of trigonometric functions, a combination of techniques using lookup tables (LUTs) and Taylor series was used. To perform the necessary calculations, a finite-state machine model (FSM) was used to reduce the use of hardware resources, however, the use of such FSM generated a sequential computation of the robotic manipulation algorithms. In this model, the forward kinematics implementation achieved a runtime of 680 ns and the inverse 940 ns, that is, for the 50 MHz clock, the forward kinematics took 34 clock cycles and the inverse kinematics took 47 cycles. Using such approaches to reduce the use of hardware resources increases computation runtime. For tactile device applications, it is important to optimize the runtime rather than the use of hardware resources.

Similarly, an FPGA implementation of forward and inverse kinematics for a 7-DoF device was presented in (Jiang et al. 2017), however, only 3-DoF required to control the device movement were implemented in hardware. The proposal used a 32-bit fixed-point representation and a CORDIC was used to execute the trigonometric functions. To validate the proposal, the FPGA was set to receive the three reference angles, perform the forward kinematics and then the inverse. The model was developed based on pipeline and the operating frequency used was of 100 MHz. As a result, the model calculation took 2μs to perform the entire kinematics algorithm, which represented 200 clock cycles.

In this context, it is possible to realize that the use of reconfigurable FPGA-based computing can accelerate haptic device control algorithms. Unlike traditional hardware that processes information sequentially, FPGA enables parallel information processing. However, most studies from the literature have developed partially parallel implementations, that is, implementations in which part of the used algorithms is executed sequentially. Unlike the researches previously mentioned, this study presents a new approach in which the execution of the robotic manipulation algorithms are performed in a full-parallel hardware implementation. This proposed implementation provides a latency reduction for the tactile devices and enables tactile internet applications.

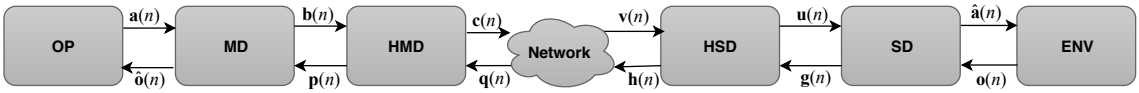


Figure 2.1: Proposed discrete model of a tactile internet system.

2.3 Discrete Model of Tactile Internet

A discrete model of a tactile internet system is proposed and presented in Figure 2.1. This model consists of seven subsystems called: Operator (OP), Master Device (MD), Hardware of the MD (HMD), Network (NW), Hardware of the SD (HSD), Slave Device (SD) and Environment (ENV) and it is assumes that the signals are sampled at a time t_s.

The OP is an entity responsible for generating stimuli that can be in the form of position signals, speed, force, image, sound or any other. These stimuli are sent to the devices involved to that some kind of task can be performed in some kind of environment. The environment, ENV subsystem, receives the stimuli from the OP and generates feedback signals associated with sensations such as reaction force information and tactile information that are sent back to the OP. The interaction between the OP and the ENV is performed through the master and slave devices, MD and SD, respectively.

Specifically in this work, MD is characterized as a local device, SD as remote one and both of them are responsible for transforming the stimuli and sensations associated with OP and ENV into signals to be processed. Tactile devices (MD and SD) can take the form of robotic manipulators, haptic devices, tactile gloves and others that may be developed in the future. In the coming years, it is expected the introduction of new types of sensors and actuators that will form the basis for the development of new tactile devices.

Although there are no tactile internet standards nor products yet, it can be affirmed that future tactile devices will be integrated with a hardware responsible for all operational metrics and calculations. Within this conjecture, this work adds a couple of modules to the discrete model (as per Figure 2.1), called HMD and HSD. HMD is responsible for performing all transformations and calculations associated with MD, and HSD performs the equivalent operations for the SD. Several algorithms associated with transformation, compression, control, prediction will be under the responsibility of these two modules.

Based on the model presented in Figure 2.1, the signals generated by the OP can be characterized by the array $\mathbf{a}(n)$ expressed as

$$\mathbf{a}(n) = [a_1(n), \ldots, a_i(n), \ldots, a_{N_{OP}}(n)] \tag{2.1}$$

where $a_i(n)$ is the i-th stimulus at the n-th instant and N_{OP} is the total number of stimuli signals generated by the OP. At every n-th moment the stimulus array, $\mathbf{a}(n)$, is received by the MD which transforms the stimuli into a set of N_{MD} signals expressed as

$$\mathbf{b}(n) = [b_1(n), \ldots, b_i(n), \ldots, b_{N_{MD}}(n)] \tag{2.2}$$

where $b_i(n)$ is the i-th signal generated by MD at the n-th instant. It can be stated that

at each n-th moment a set of stimuli $\mathbf{a}(n)$ generates a set of signals $\mathbf{b}(n)$ that depends on the type of MD and the sensor set associated with the device. Especially important is the fact that the signals generated by MD, $\mathbf{b}(n)$, have heterogeneous characteristics in which each i-th signal $b_i(n)$ can represent an angle, spatial coordinate, pixel of an image, audio sample or any other information associated with a stimulus generated by OP. In practice the signals grouped by the $\mathbf{b}(n)$ array originate from sensors coupled to the MD and the amount of data may vary according to the amount of information to be sent, N_{MD}.

The set of signals, expressed by $\mathbf{b}(n)$ are sent to the HMD (Figure 2.1) which has the function of processing this information before sending it to the NW subsystem. Calculations associated with calibration, linear and nonlinear transformations and signal compression are performed by the HMD. Essentially the majority of the computational effort of MD is in this subsystem. At each n-th instant t_s the HMD processes the array $\mathbf{b}(n)$ generating an information array $\mathbf{c}(n)$ expressed by

$$\mathbf{c}(n) = \left[c_1(n), \ldots, c_i(n), \ldots, c_{N^f_{HMD}}(n)\right] \tag{2.3}$$

where $c_i(n)$ is the i-th signal generated by HMD towards the subsystem NW at the n-th instant t_s and N^f_{HMD} is the numbers of signals. $N^f_{HMD} < N_{MD}$ is expected to minimize latency during the transmission in the NW subsystem.

The NW subsystem, as shown in Figure 2.1, characterizes the communication medium that links OP to ENV. In this model, the data propagates through two different channels called the forward channel, that transmits the OP data towards the ENV, and the backwards channel, that transmits the ENV signals towards the OP. The signal transmitted by the forward and backwards channels may be disturbed and delayed. In the case of the forward channel, the received signal, $\mathbf{v}(n)$, may be expressed as

$$\mathbf{v}(n) = \left[v_1(n), \ldots, v_i(n), \ldots, v_{N^f_{HMD}}(n)\right] \tag{2.4}$$

where

$$v_i(n) = c_i\left(n - d^f_i(n)\right) + r^f_i(n) \tag{2.5}$$

in which, $r^f_i(n)$ represents the added noise and $d^f_i(n)$ represents a delays associated with the i-th information sent in $\mathbf{c}(n)$. In this model the noise can be characterized as a random Gaussian variable of zero mean and σ^2_{rf} variance and the delays are characterized as integers, that is, they occur at a granularity of t_s. It is important to note that the NW subsystem can take the shape of the Internet, a metropolitan network (MAN), a local area network (LAN), or even a direct connection between an MD and a workstation or computer.

As shown in Figure 2.1, the HSD receives the $\mathbf{v}(n)$ signal through the forward channel and has the role of generating control signals to the SD through the signal

$$\mathbf{u}(n) = \left[u_1(n), \ldots, u_i(n), \ldots, u_{N^f_{HSD}}(n)\right] \tag{2.6}$$

where N^f_{HSD} is the number of control signals and $u_i(n)$ is i-th control signal at the n-th

instant t_s associated with the array $\mathbf{u}(n)$. It is important to note that there may be various types of SD: from real robotic handlers to virtual tools in computational environments. Thus, it can be stated without loss of generality that HSD can perform an inverse processing to HMD in addition to specific algorithms associated with the type of SD. For example, if the SD is a robotic handler, HSD must additionally implement closed loop control algorithms, whereas if SD is a virtual arm HSD must implement positioning algorithms for a given virtual reality platform. SD does not have to correspond directly with MD, e.g. MD can be a glove while SD is a drone. However, it is desirable that the stimulus generated by the SD is a copy of the stimulus generated by the OP, that is, within the model presented in Figure 2.1, it can be understood that SD generate a signal expressed as

$$\hat{\mathbf{a}}(n) = [\hat{a}_1(n), \ldots, \hat{a}_i(n), \ldots, \hat{a}_{N_{OP}}(n)] \tag{2.7}$$

where $\hat{a}_i(n)$ is an estimate of the i-th stimulus $a_i(n)$ generated by the OP. Thus, the estimate of the stimulus generated by OP, $\hat{a}_i(n)$, is applied to the ENV subsystem representing a given real or virtual environment in which OP is interacting.

In the backwards direction, the stimulus actions generated by OP, $\mathbf{a}(n)$, and represented by $\hat{\mathbf{a}}(n)$, receives a group of reactions from the ENV subsystem that can be characterized in the model by the set of signals expressed by

$$\mathbf{o}(n) = [o_1(n), \ldots, o_i(n), \ldots, o_{N_{ENV}}(n)] \tag{2.8}$$

where N_{ENV} is the number of stimulus signals and $o_i(n)$ is i-th stimulus signal at the n-th instant t_s. Reaction signals grouped into $\mathbf{o}(n)$ can be in the form of strength, touch, temperature, etc.

Reaction signals are captured by the SD that turns this information into electrical signals from real or virtual sensors, if the SD is in a virtual reality environment. After capturing this information the SD transmits these signals to the HSD. In the model presented in Figure 2.1, the signals generated by the SD are expressed as

$$\mathbf{g}(n) = [g_1(n), \ldots, g_i(n), \ldots, g_{N_{SD}}(n)] \tag{2.9}$$

where $g_i(n)$ is the i-th signal generated by the SD at the n-th instant of time, t_s and N_{SD} is the amount of signals. The HSD in turn processes this information and sends to the NW subsystem through the array $\mathbf{h}(n)$, expressed by

$$\mathbf{h}(n) = \left[h_1(n), \ldots, h_i(n), \ldots, h_{N^b_{HSD}}(n)\right] \tag{2.10}$$

where $h_i(n)$ is the i-th signal generated by HSD at the n-th instant of time, t_s and N^b_{HSD} is the amount of signals.

The signal received by the HMD through the backwards channel of the NW subsystem can be expressed as

$$\mathbf{q}(n) = \left[q_1(n), \ldots, q_i(n), \ldots, q_{N^b_{HSD}}(n)\right] \tag{2.11}$$

where

$$q_i(n) = h_i\left(n - d_i^b(n)\right) + r_i^b(n) \tag{2.12}$$

in which, $r_i^b(n)$ represents an added noise and $d_i^b(n)$ represents a delay associated with the i-th information transmitted in $\mathbf{q}(n)$ by the backwards channel. Similarly to the forward channel, noise can also be characterized as a random variable Gaussian of zero mean and variance σ_{rb}^2 and delays are characterized as integers with t_s granularity. The HMD processes the $\mathbf{q}(n)$ signal information and generates a set of control signals that will act on the MD and can be characterized as

$$\mathbf{p}(n) = \left[p_1(n), \ldots, p_i(n), \ldots, p_{N_{HMD}^b}(n)\right] \tag{2.13}$$

where $p_i(n)$ is the i-th signal generated by the HMD at the n-th instant of time t_s and N_{HMD}^b is the number of signals. The MD in turn will synthesize the reaction stimuli generated by the environment, i.e. the ENV subsystem. Based on the model, it is possible to characterize these reaction stimuli as a signal expressed by

$$\hat{\mathbf{o}}(n) = \left[\hat{o}_1(n), \ldots, \hat{o}_i(n), \ldots, \hat{o}_{N_{ENV}}(n)\right]. \tag{2.14}$$

where $\hat{o}_i(n)$ is an estimate of the i-th stimulus $o_i(n)$ generated in the ENV subsystem. Examples of reaction stimuli generated or synthesized by MD are touch, strength and temperature.

In addition to the latency associated with the NW subsystem that characterizes the communication medium between the OP and ENV subsystems, the MD, HMD, HSD, and SD subsystems also add latency to the system. Based on the work presented in (Szabo, Gulyas, Fitzek, Fitzek & Lucani 2015, Dohler et al. 2017) these components represent 30% of total latency. The latency of the MD and SD subsystems are associated with sensors and actuators that can be mechanical, electrical, electromechanical and other variations. HMD and HSD latencies are associated with the processing time of the algorithms in these devices and depending on the type of hardware and implementation architecture this latency can be considerably reduced.

2.4 PHANToM Omni Device Model (MD & SD)

Based on the scheme presented in Figure 2.1, this section presents details associated with the MD and SD used as reference for the hardware system proposed in this research. The MD and SD are characterized as a three degree of freedom robotic manipulator, 3-DoF, called the PHANToM Omni (Geomagic n.d.) (Figure 2.2). The PHANToM Omni has been widely used in literature as presented in (Song et al. 2006) and (Sansanayuth et al. 2012). In this work two of this devices are going to be used: one as an MD and the other as a SD.

As can be seen from Figure 2.3, the PHANToM Omni physical structure is formed by a base, an arm with two segments L_1 and L_2 which are interconnected by three rotary joints θ_1, θ_2 and θ_3 and a tool. The variables presented in Figure 2.3 are represented by:

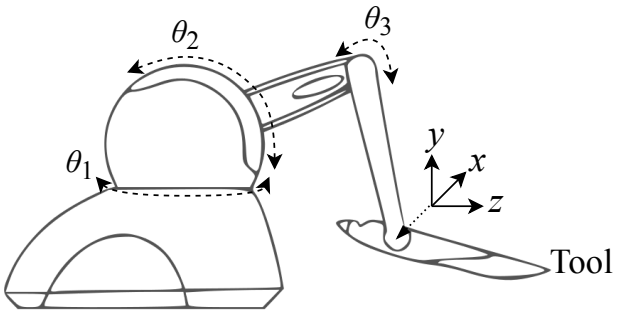


Figure 2.2: PHANToM Omni - MD and SD.

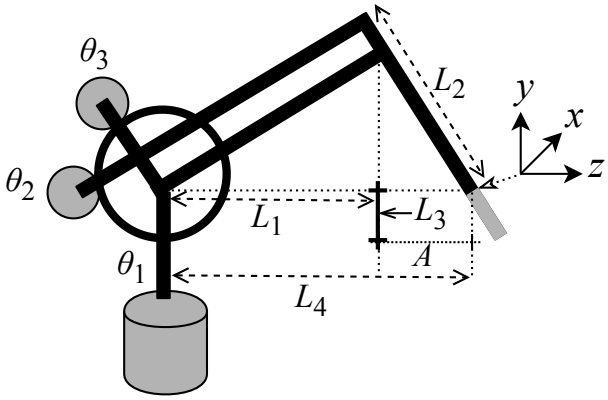


Figure 2.3: PHANToM Omni structure - MD and SD.

L_1 = 0.135mm, $L_2 = L_1$, L_3=0.025mm and $L_4 = L_1 + A$ where A=0.035mm as described in (Silva et al. 2009). These detailed features of the device are essential for performing the kinematics and dynamic calculations.

2.4.1 Forward Kinematics

The kinematics of manipulative devices makes use of the relationship between operational coordinates and joint coordinates. Forward kinematics (FK) correlates the angular variables of the joints with the Cartesian system. That is, given an array of joint coordinates it is possible to determine the spatial position of the tool through the equation that can be expressed by

$$x = -\sin(\theta_1)(L_2\sin(\theta_3) + L_1\cos(\theta_2)), \tag{2.15}$$

$$y = -L_2\cos(\theta_3) + L_1\sin(\theta_2) + L_3, \tag{2.16}$$

$$\begin{aligned} z = {} & L_2\cos(\theta_1)\sin(\theta_3) \\ & + L_1\cos(\theta_1)\cos(\theta_2) - L_4 \end{aligned} \tag{2.17}$$

where x, y and z are variables that determine the spatial position of the tool in the Cartesian plane.

2.4.2 Inverse Kinematics

In the inverse kinematics (IK), the relationship between the joint angles and the Cartesian system is reversed, that is, given the spatial position of the tool it may be possible to determine the joint coordinates. The solution to this process is not as straightforward as in the direct kinematics. In direct kinematics, the position of the tool is determined solely by the displacements of the joints. In inverse kinematics, equations are composed of nonlinear calculations formed by trigonometric functions. Depending on the manipulator structure, multiple solutions may be possible for the same tool position, or there may be no solution for a particular set of tool positions. Based on the works (Cavusoglu & Feygin 2001), (San Martin & Triviño 2006) and (Silva et al. 2009), the value of θ_1 can be defined through the equation expressed by

$$\theta_1 = -\text{atan2}\,(x, z + L_4) \tag{2.18}$$

where x and z represent coordinates in the Cartesian plane and L_4 corresponds to the size of the the arm segments, as shown in Figure 2.3.

To calculate the other two joints θ_2 and θ_3 it is necessary to perform intermediate calculations. Thus, one can obtain R, r, β, γ and α through the equations

$$R = \sqrt{x^2 + (z + L_4)^2}, \tag{2.19}$$

$$r = \sqrt{x^2 + z + L_4)^2 + (y - L_3)^2}, \tag{2.20}$$

$$\gamma = \text{acos}\left(\frac{L_1^2 - L_2^2 + r^2}{2L_1 r}\right), \tag{2.21}$$

$$\beta(n) = \text{atan2}(y - L_3, R), \tag{2.22}$$

and

$$\alpha = \text{acos}\left(\frac{L_1^2 + L_2^2 - r^2}{2L_1 L_2}\right). \tag{2.23}$$

After performing the intermediate calculations it is possible to calculate θ_2 through the equation

$$\theta_2 = \gamma + \beta. \tag{2.24}$$

Finally, the value corresponding to the θ_3 joint can be obtained through the equation

$$\theta_3 = \theta_2 + \alpha - \frac{\pi}{2}. \tag{2.25}$$

2.4.3 Kinesthetic Feedback Force

The kinesthetic feedback force allows the environment to be "felt", i.e. when the SD comes into physical contact with an object, the MD will receive a counter force. This model can be implemented through the equation

$$\tau = \mathbf{J}^T \mathbf{F}, \tag{2.26}$$

where τ defines the torque array that will be applied to each joint (θ_1, θ_2 and θ_3) of the PHANToM Omni associated with the MD, $\mathbf{J}^T$ is the transpose of the Jacobian matrix and $\mathbf{F}$ is the force array resulting from the interaction of SD with ENV. The torque array τ can be expressed as

$$\tau = [\tau_1, \tau_2, \tau_3]. \tag{2.27}$$

The $\mathbf{J}$ Jacobian matrix incorporates structural information about the handler and it is identified as

$$\mathbf{J} = \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix}, \tag{2.28}$$

where

$$J_{11} = -\cos(\theta_1)(L_2 \sin(\theta_3) + L_1 \cos(\theta_2)), \tag{2.29}$$

$$J_{21} = 0, \tag{2.30}$$

$$J_{31} = -L_1 \cos(\theta_2)\sin(\theta_1) - L_2 \sin(\theta_3)\sin(\theta_1), \tag{2.31}$$

$$J_{12} = L_1 \sin(\theta_1)\sin(\theta_2), \tag{2.32}$$

$$J_{22} = L_1 \cos(\theta_2), \tag{2.33}$$

$$J_{32} = -L_1 \sin(\theta_2)\cos(\theta_1), \tag{2.34}$$

$$J_{13} = -L_2 \sin(\theta_1)\cos(\theta_3), \tag{2.35}$$

$$J_{23} = L_2 \sin(\theta_3), \tag{2.36}$$

and

$$J_{33} = L_2 \cos(\theta_3)\cos(\theta_1). \tag{2.37}$$

The force array $\mathbf{F}$ is expressed as

$$\mathbf{F} = [F_x, F_y, F_z] \tag{2.38}$$

and can be obtained through sensors internal or external to the device. According to Equation 2.26, the τ torque array representing the resulting force at each joint can be defined as

$$\tau_1 = J_{11}F_x + J_{21}F_y + J_{31}F_z, \tag{2.39}$$

$$\tau_2 = J_{12}F_x + J_{22}F_y + J_{32}F_z, \tag{2.40}$$

and

$$\tau_3 = J_{13}F_x + J_{23}F_y + J_{33}F_z. \tag{2.41}$$

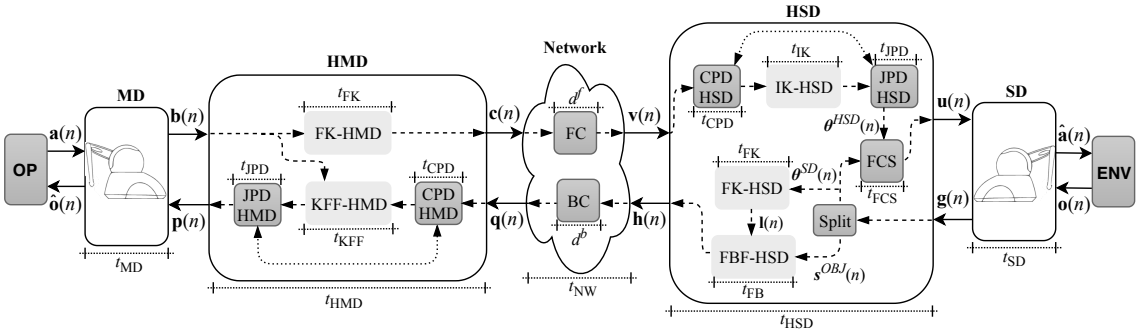


Figure 2.4: Detailed discrete model of a tactile internet system.

2.5 Simulated Tactile Internet Model

Figures 2.1 and 2.4 details the structure used for the hardware design in FPGA, in which a given operator, OP, handles a PHANToM Omni on the master side, MD, which is connected to HMD that, in this case, is a dedicated FPGA hardware. Data is transmitted through the network, the NW subsystem, to HSD which is also a dedicated hardware in FPGA. The HSD is also connected to a PHANToM Omni that interacts with the environment, the ENV subsystem. Figure 2.4 also details the backwards direction from the ENV and the OP.

The OP is modeled as an information source responsible for generating a spatial trajectory through discrete signals expressed in the $\mathbf{a}(n)$ array. At each n-th instant t_s the OP sends three variables $x^{OP}(n)$, $y^{OP}(n)$ and $z^{OP}(n)$ representing the positioning of the MD tool (Figures 2.2 and 2.3) in the Cartesian space an this is expressed by

$$\mathbf{a}(n) = \left[x^{OP}(n), y^{OP}(n), z^{OP}(n)\right]. \tag{2.42}$$

This step simulates the spatial movement of the MD tool by the operator, that is, at each instant of time, t_s, a spatial movement is performed and a new signal $\mathbf{a}(n)$ is generated by the OP.

The PHANToM Omni has encoders at its three joints that translate spatial positioning at the three angles θ_1, θ_2 and θ_3 (Figures 2.2 and 2.3). Thus, based on Figure 2.4, it can be said that MD converts the signal $\mathbf{a}(n)$ into a signal expressed as

$$\mathbf{b}(n) = \left[\theta_1^{MD}(n), \theta_2^{MD}(n), \theta_3^{MD}(n)\right] \tag{2.43}$$

and forwards it to the HMD at every n-th instant of time t_s.

Then, as can be seen in Figure 2.4, the $\mathbf{b}(n)$ signal propagates to the HMD, which on receiving the signal transforms the joint positioning angles, $\mathbf{b}(n)$, into spatial position by calculating the FK according to the Equations 2.15, 2.16 and 2.17. All equations are implemented in FPGA through a hardware module called the FK-HMD. The equations are implemented in parallel which can significantly increase the processing time. The use of FK is motivated by a reduction of the amount of information utilized, i.e., for a N-

DoF robotic manipulator N joint angles will be generated and that can be converted into only three values associated with the spatial position of the tool, x, y and z. On the other hand, the use of this strategy increases the amount of calculations to be performed by the MD, which is compensated by the parallel implementation of the algorithm in FPGA. It is essential to note that the use of custom hardware operating in parallel allows processing time not to be substantially affected by N.

Based on Section 2.3, after the FK calculation by the FK-HMD hardware module, a new discrete signal is created that can be expressed by

$$\mathbf{c}(n) = \left[x^{HMD}(n), y^{HMD}(n), z^{HMD}(n)\right] \tag{2.44}$$

where $x^{HMD}(n)$, $y^{HMD}(n)$ and $z^{HMD}(n)$ are the values of the spatial coordinate array generated by the HMD to be sent to HSD via the communication medium, NW. The FK-HMD hardware module generates a new $\mathbf{c}(n)$ array every n-th instant of time.

After the transmission through the forward channel, here called FC, the signal received by the HSD can be expressed as

$$\mathbf{v}(n) = \left[x^{HSD}(n), y^{HSD}(n), z^{HSD}(n)\right]. \tag{2.45}$$

Based on equation 2.5 the spatial coordinate signal received by HSD can be expressed as

$$x^{HSD}(n) = x^{HMD}\left(n - d_x^f(n)\right) + r_x^f(n), \tag{2.46}$$

$$y^{HSD}(n) = y^{HMD}\left(n - d_y^f(n)\right) + r_y^f(n), \tag{2.47}$$

and

$$z^{HSD}(n) = z^{HMD}\left(n - d_z^f(n)\right) + r_z^f(n) \tag{2.48}$$

where $d_x^f(n)$, $d_y^f(n)$, $d_z^f(n)$, $r_x^f(n)$, $r_y^f(n)$ and $r_z^f(n)$ are the delays and noises associated with CF.

As in this case the Slave PHANToM Omni, SD, copies the movement of the master PHANToM Omni, MD, it is necessary for the HSD to perform a feedback control system on the three joints of the PHANToM Omni slave, here expressed as

$$\theta^{SD}(n) = \left[\theta_1^{SD}(n), \theta_2^{SD}(n), \theta_3^{SD}(n)\right] \tag{2.49}$$

that is, $\theta_1^{SD}(n)$, $\theta_2^{SD}(n)$, $\theta_3^{SD}(n)$ are control variables associated with DS. The control system illustrated in Figure 2.4 as FCS shall minimize the error, $\mathbf{e}^{FCS}(n)$, between $\theta^{SD}(n)$ and the reference signal $\theta^{HSD}(n)$ characterized as

$$\theta^{HSD}(n) = \left[\theta_1^{HSD}(n), \theta_2^{HSD}(n), \theta_3^{HSD}(n)\right] \tag{2.50}$$

where

$$\mathbf{e}(n) = \theta^{HSD}(n) - \theta^{SD}(n) and \tag{2.51}$$

and

$$\begin{bmatrix} e_1^{FCS}(n) \\ e_2^{FCS}(n) \\ e_3^{FCS}(n) \end{bmatrix} = \begin{bmatrix} \theta_1^{HSD}(n) \\ \theta_2^{HSD}(n) \\ \theta_3^{HSD}(n) \end{bmatrix} - \begin{bmatrix} \theta_1^{SD}(n) \\ \theta_2^{SD}(n) \\ \theta_3^{SD}(n) \end{bmatrix}. \tag{2.52}$$

The $\theta^{SD}(n)$ signal is obtained from the SD via sensors (encoders) at the SD joints and the $\theta^{HSD}(n)$ signal is obtained from the IK-HSD hardware module shown in Figure 2.4. This hardware module implements all inverse kinematics equations presented in Section 2.4.2, i.e. Equations 2.18 through 2.25. There are several techniques and approaches that can be used in the FCS module ranging from more traditional techniques such as a proportional–integral–derivative controller (Kumar et al. 2017) to more innovative artificial intelligence based techniques (Yang et al. 2016, Rahimi & Nazemizadeh 2014).

The CPD-HSD and JPD-HSD modules, illustrated in Figure 2.4, represent the algorithms of prediction and detection in cartesian space and joints, respectively. These modules are responsible for minimizing the latency and noise added by the FC associated with the tactile internet system (Equations 2.46, 2.47 and 2.48). Depending on the prediction and detection technique used, the HSD may use only one of the modules, namely the CPD-HSD or JPD-HSD. There is still no consensus about whether the Cartesian space or joints is the best for minimizing latency and noise inserted by the channel. There are several works in the literature that present proposals using only one of the spaces and proposals that try to use the information from both simultaneously.

Similarly to the FCS module, approaches ranging from the more traditional techniques up to more innovative techniques based on artificial intelligence have been used in the CPD-HSD and JPD-HSD modules (Tang et al. 2014, Chen & Li 2018, Xiang 2019, Bócsi et al. 2011, Shen et al. 2019). Thus, it can be said that $\theta^{HSD}(n)$ is an estimate of the $\mathbf{b}(n)$ signal generated by the MD.

At each n-th time, the FCS acts on the SD through the $\mathbf{u}(n)$ signal, detailed in Figures 2.1 and 2.4, which in the case of the PHANToM Omni can be expressed as

$$\mathbf{u}^{HSD}(n) = \left[\tau_1^{HSD}(n), \tau_2^{HSD}(n), \tau_3^{HSD}(n)\right] \tag{2.53}$$

where $\tau_i^{HSD}(n)$ is the i-th torque applied every i-th joint. The FCS will act as a tracking mechanism, making the SD follow the path traveled by the MD. Finalizing the data stream associated with the forward channel, it can be said that the $\hat{\mathbf{a}}(n)$ signal is formed by an estimate of the spatial position generated by the OP, $\hat{\mathbf{a}}(n)$, i.e.

$$\hat{\mathbf{a}}(n) = \left[\hat{x}^{OP}(n), \hat{y}^{OP}(n), \hat{z}^{OP}(n)\right]. \tag{2.54}$$

The interaction of the PHANToM Omni, SD, with ENV can vary from free movement to physical contact. When some kind of physical contact occurs, the SD detects the touch and sends this information back to the HSD. As per the model detailed in Figure 2.4 the ENV sends back to SD the information associated with the contact force in the spatial plane, expressed here as,

$$\mathbf{o}(n) = \left[F_x^{ENV}(n), F_y^{ENV}(n), F_z^{ENV}(n)\right]. \tag{2.55}$$

The value associated with the contact force information can be measured directly through SD-coupled force sensors or indirectly estimated through other types of sensors that may be SD-coupled or inserted into the environment (Yang et al. 2018*a*). In the case of the model presented in Figure 2.4, the SD sends to HSD the objects surface's spatial positions through sensors spread in the ENV. The signal expressed as

$$\mathbf{s}^{OBJ}(n) = \left[x^{OBJ}(n), y^{OBJ}(n), z^{OBJ}(n)\right] \tag{2.56}$$

represents the spatial position of the closest object from the SD tool. Thus, based on the information already described, every n-th time t_s the SD sends to the HSD a signal characterized by the array $\mathbf{g}(n)$ expressed as

$$\mathbf{g}(n) = \left[\theta^{SD}(n), \mathbf{s}^{OBJ}(n)\right]. \tag{2.57}$$

In the HSD, when the signal $\mathbf{g}(n)$ is received, the Split module separates the $\theta^{SD}(n)$ signal and sends it to the FCS and the FK-HSD hardware module. And the signal $\mathbf{s}^{OBJ}(n)$ is sent to the FB-HSD hardware module, as detailed in Figure 2.4. The FK-HSD hardware module performs the forward kinematics calculation similarly to FK-HMD and thus the current spatial position of the SD tool in the environment, ENV, can be obtained. Every n-th instant t_s FK-HSD generates a signal expressed as

$$\mathbf{l}(n) = [x^{ENV}(n), y^{ENV}(n), z^{ENV}(n)] \tag{2.58}$$

where $x^{ENV}(n)$, $y^{ENV}(n)$ and $z^{ENV}(n)$ are the spatial position of the tool in the ENV module from $\theta^{SD}(n)$. The FBF-HSD hardware module implements the calculations associated with the generation of the feedback force from the contact between the tool and the object. Based on the work presented in (Yang et al. 2018*a*) the contact force, represented by the $\mathbf{h}(n)$ signal, can be expressed as

$$\mathbf{h}(n) = \left[F_x^{HSD}(n), F_y^{HSD}(n), F_z^{HSD}(n)\right], \tag{2.59}$$

where

$$F_x^{HSD}(n) = h_x(n)\left(x^{OBJ}(n) - x^{ENV}(n)\right), \tag{2.60}$$

$$F_y^{HSD}(n) = h_y(n)\left(y^{OBJ}(n) - y^{ENV}(n)\right), \tag{2.61}$$

and

$$F_z^{HSD}(n) = h_z(n)\left(z^{OBJ}(n) - z^{ENV}(n)\right). \tag{2.62}$$

In these equations, the constants $h_x(n)$, $h_y(n)$ and $h_z(n)$ represent the elasticity coefficients associated with the object. It is important to note that in this model the $\mathbf{h}(n)$ signal is a synthesized version of the real force value here characterized by the $\mathbf{o}(n)$ array.

After the feedback force calculation process, as illustrated in Figure 2.4, the $\mathbf{h}(n)$ signal is transmitted to the HMD via the backwards channel (BC) which, similarly to FC,

adds latency and noise. The signal received by the HMD can be expressed as

$$\mathbf{q}(n) = [F_x^{HMD}(n), F_y^{HMD}(n), F_z^{HMD}(n)] \tag{2.63}$$

where

$$F_x^{HMD}(n) = F_x^{HSD}\left(n - d_x^b(n)\right) + r_x^b(n), \tag{2.64}$$

$$F_y^{HMD}(n) = F_y^{HSD}\left(n - d_y^b(n)\right) + r_y^b(n), \tag{2.65}$$

and

$$F_z^{HMD}(n) = F_z^{HSD}\left(n - d_z^b(n)\right) + r_z^b(n) \tag{2.66}$$

where $d_x^b(n)$, $d_y^b(n)$, $d_z^b(n)$, $r_x^b(n)$, $r_y^b(n)$ and $r_z^b(n)$ are the latencies and the noises associated with the BC.

Similarly to HSD, the HMD will minimize the effect of latency and noise from operations of Cartesian and joint space. For HMD, the calculations associated with the Cartesian space will be performed by the CPD-HMD module and associated with the joint space by the JPD-HMD module. In addition to the prediction and detection calculations, the HMD must transform the force signals received through signal $\mathbf{q}(n)$ into a torque to be applied to the MD joints which is accomplished by the KFF-HMD hardware module. KFF-HMD implements the Equations 2.39, 2.40 and 2.41 presented in Section 2.4.3 and generate the signal expressed as

$$\mathbf{p}(n) = \left[\tau_1^{HMD}(n), \tau_2^{HMD}(n), \tau_3^{HMD}(n)\right] \tag{2.67}$$

where $\tau_i^{HMD}(n)$ is the torque associated with the i-th joint of the MD. Since the PHANToM Omni is a haptic device, it already has a built-in control system, FCS, which uses as reference signal the torques associated with the $\mathbf{p}(n)$ array.

After applying the torques to the MD joints via the $\mathbf{p}(n)$ signal, the OP receives the feedback force signal, in other words, it feels the object touched by the SD in the ENV. This sensation is identified in by the $\hat{\mathbf{o}}(n)$ signal expressed as

$$\hat{\mathbf{o}}(n) = \left[\hat{F}_x^{ENV}(n), \hat{F}_y^{ENV}(n), \hat{F}_z^{ENV}(n)\right]. \tag{2.68}$$

As illustrated in Figure 2.4, the MD, HMD, NW, HSD, and SD subsystems have the following runtimes: t_{MD}, t_{HMD}, t_{NW}, t_{HSD} and t_{SD}, respectively. The sum of these, times taking into account the forward direction (between OP and ENV) and the backwards direction (between ENV and OP), represent the total system latency that can be expressed as

$$t_{\text{latency}} = 2\left(t_{\text{MD}} + t_{\text{HMD}} + t_{\text{NW}} + t_{\text{HSD}} + t_{\text{SD}}\right). \tag{2.69}$$

Some works presented in the literature review agree that the ideal requirement is that $t_{\text{latency}} \leq 1\,\text{ms}$, on the other hand, other works point out that the latency requirement can be expresses as $t_{\text{latency}} \leq 10\,\text{ms}$, depending on the application (Li et al. 2018, Antonakoglou et al. 2018*a*, Nasrallah et al. 2018, Simsek et al. 2016*c*, Junior et al. 2019). Considering that 30% of the total latency time t_{latency} is spent by MD, HMD, HSD, and SD, it can be

understood that

$$(t_{MD} + t_{HMD} + t_{HSD} + t_{SD}) \leq \frac{0.3t_{latency}}{2}. \tag{2.70}$$

Assuming an equal time division among MD, HMD, HSD, and SD it is possible to affirm that the time associated with hardware, $t_{hardware}$, whether the master, HMD, or the slave device, HSD, can be expressed as

$$t_{HMD} = t_{HSD} = t_{hardware} \leq \frac{0.3t_{latency}}{8}. \tag{2.71}$$

Taking the 1 ms constraints into consideration and substituting this value in Equation 2.71, it is possible to affirm that the hardware time, $t_{hardware}$, must meet the $t_{hardware} \leq 37.5\,\mu s$ constraint for all cases (condition 1 ms) or the $t_{hardware} \leq 375\,\mu s$ constraint for some specific cases (10 ms condition). This equation can be improved, it is just an initial proposal.

Recent studies from the literature show that the 1 ms restriction ($t_{hardware} \leq 37.5\,\mu s$) is difficult to achieve using hardware devices based on embedded systems such as microprocessors and microcontrollers (Weber et al. 2016, Arjun et al. 2018). The 10 ms restriction ($t_{hardware} \leq 375\,\mu s$) is achieved in specific cases where SD is a virtual environment and HSD is a high performance processor computer (Junior et al. 2019). Thus this work aims to minimize the execution time in HMD, t_{HMD}, and HSD, t_{HSD}, using FPGA reconfigurable computation. In other words, the target is to achieve a $t_{hardware} \leq 37.5\,\mu s$.

This work presents a hardware reference model for the FK-HMD, KFF-HMD, IK-HSD, FK-HSD, and FBF-HSD modules illustrated in Figure 2.4. The complete model that will be presented in detail in the next section makes use of a parallel implementation methodology in which high throughput is prioritized, i.e. the execution time of the modules t_{FK}, t_{KFF}, t_{IK} and t_{FBF}, illustrated in Figure 2.4.

This work does not propose dedicated hardware reference models for the CPD-HSD, JPD-HSD, CPD-HMD, JPD-HMD and FCS modules as there are several techniques and algorithms that can be applied to them. However, considering the hardware time constraints, $t_{hardware}$, it is noted that it is also important to use dedicated hardware structures with reconfigurable computing for these modules. Studies in the literature foresee the use of AI based techniques for these modules however it is essential to note that AI techniques and algorithms implemented on general purpose processor-based hardware platforms can lead to higher processing time. (de Souza & Fernandes 2014, Da Costa et al. 2019, Coutinho et al. 2019, Torquato & Fernandes 2019, Da Silva et al. 2019, Lopes et al. 2019, Noronha et al. 2019).

2.6 Implementation Description

The FK-HMD and KFF-HMD hardware modules associated with the master device (HMD) and the IK-HSD, FK-HSD, and FBF-HSD hardware modules associated with the slave device (HSD) (Figure 2.4) were designed using a parallel implementation in order to prioritize the processing speed. The implementations were designed in FPGA using a hybrid scheme with fixed-point and floating-point representation in distinct parts of the

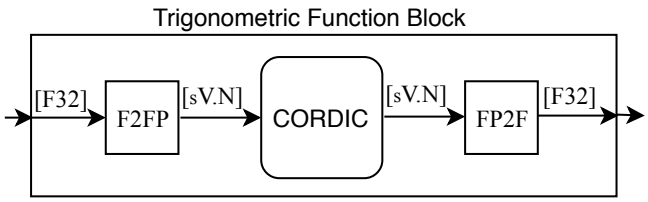


Figure 2.5: Proposed circuit for calculating trigonometric functions - TFB.

proposed architecture. In the portions that adopts the fixed-point format, the variables follow a notation expressed as $[sV.N]$ indicating that the variable is formed by V bits of which N bits are intended for the fractional part and the s symbol indicates that the variable is signed. In this case, the number of bits intended for the integer part is $V - N - 1$. For the representation of floating-point variables, the notation [F32] is adopted. Most of the implemented circuits were designed using a 32-bit single precision (IEEE754) floating-point format representation. The fixed-point format was used only on the circuit that implements the trigonometric function block (TFB) module, as illustrated in Figure 2.5. TFB is the module responsible for performing trigonometric operations through the hardware implementation of CORDIC (*COordinate Rotation DIgital Computer*) (Volder 1959). The implemented CORDIC circuit uses data representation in fixed-point format using the $[s16.13]$ representation.

As illustrated in Figure 2.5, the TFB module receives data from external circuits in the 32-bits floating-point standard. A conversion to the fixed-point numeric representation type represented by the $[s16.13]$ notation is performed through the Float to Fixed-point (F2FP) module that has been implemented in hardware. After the CORDIC hardware operations are performed, the data in the fixed-point format is transformed back to the 32-bit floating-point through the Fixed-point to Float (FP2F) module which was also implemented in hardware.

Several of the proposed methods to be presented use the constants L_1, L_2, L_3 and L_4. They represent physical characteristics of the PHANToM Omni device as illustrated in Figure 2.2. These constants use the 32-bit floating-point numeric representation.

2.6.1 Forward Kinematics (FK-HMD & FK-HSD))

As illustrated in Figure 2.4, both the hardware associated with the master device (HMD) and the hardware associated with the slave device (HSD) implement forward kinematics through the FK-HMD and FK-HSD modules, respectively. These modules have the same FPGA-implemented circuit, differing only in the input and output signals. They are designed to work with three input signals, one for each component of the angular positioning of the device's joints, and three output signals, one for each component of the the positioning of the device's tool in the Cartesian system. The input signals are $\theta_1[F32](n)$, $\theta_2[F32](n)$ and $\theta_3[F32](n)$ and the output signals are $x[F32](n)$, $y[F32](n)$ and $z[F32](n)$. For FK-HMD, the input signals represent the $\theta_1^{MD}[F32](n)$, $\theta_2^{MD}[F32](n)$ and $\theta_3^{MD}[F32](n)$ signals, and the output signals represent the $x^{HMD}[F32](n)$, $y^{HMD}[F32](n)$

and $z^{HMD}[F32](n)$ signals. In the case of the FK-HSD module, the input signals represent the signals $\theta_1^{SD}[F32](n)$, $\theta_2^{SD}[F32](n)$ and $\theta_3^{SD}[F32](n)$ and the output signals represent the signals $x^{ENV}(n)$, $y^{ENV}(n)$ and $z^{ENV}(n)$. At every n-th instant all the computation performed in order to calculate the forward kinematics are executed in parallel.

Based on Equation 2.15, the algorithm used for calculating $x[F32](n)$ was implemented in FPGA through the generic circuit illustrated in Figure 2.6. The circuit was designed to work with three input signals $\theta_1[F32](n)$, $\theta_2[F32](n)$ and $\theta_3[F32](n)$ and one output signal. These signals are forwarded to TFB sub circuits where sine and cosine calculations are performed. For this process the constants L_1 and L_2, three multipliers, one inverter and one adder are used.

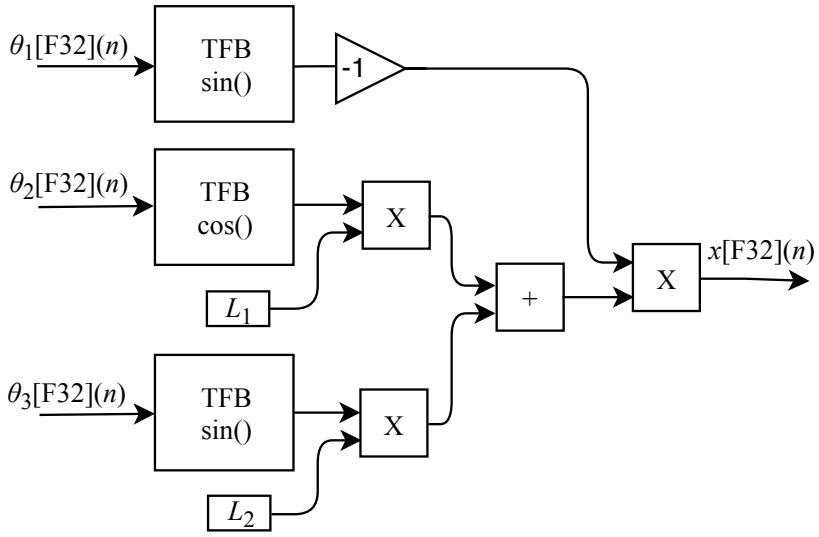


Figure 2.6: Proposed forward kinematics circuit for obtaining the $x[F32](n)$ spatial coordinate (Eq. 2.15) - FK-HMD & FK-HSD.

The calculation of $y[F32](n)$ based on Equation 2.16 was implemented in FPGA through the generic circuit shown in Figure 2.7. The circuit was designed to work with two input signals $\theta_2[F32](n)$ and $\theta_3[F32](n)$ and one output signal. These signals are routed to TFB sub circuits to perform sine and cosine calculations. In the process flow two multipliers, two adders, one inverter and the constants L_1 and L_2 are used.

The generic circuit illustrated in Figure 2.8 was implemented in FPGA to perform the calculation of $z[F32](n)$ and it is based on Equation 2.17. The circuit is designed to work with three input signals $\theta_1[F32](n)$, $\theta_2[F32](n)$ and $\theta_3[F32](n)$ and one output signal. These signals are routed to TFB sub circuits in order to perform sine and cosine calculations. In the process flow four multipliers, two adders, one inverter and the constants L_1, L_2 and L_4 are used.

In the FK-HMD module the $\theta_1^{MD}[F32](n)$, $\theta_2^{MD}[F32](n)$ and $\theta_3^{MD}[F32](n)$ input signals are received through the $\mathbf{b}(n)$ array (eq. 2.43), then all calculation are performed in parallel resulting in the $\mathbf{c}(n)$ array (eq. 2.44) with the $x^{HMD}[F32](n)$, $y^{HMD}[F32](n)$ and $z^{HMD}[F32](n)$ signals as shown in Figure 2.4. For the FK-HSD module the $\theta_1^{SD}[F32](n)$,

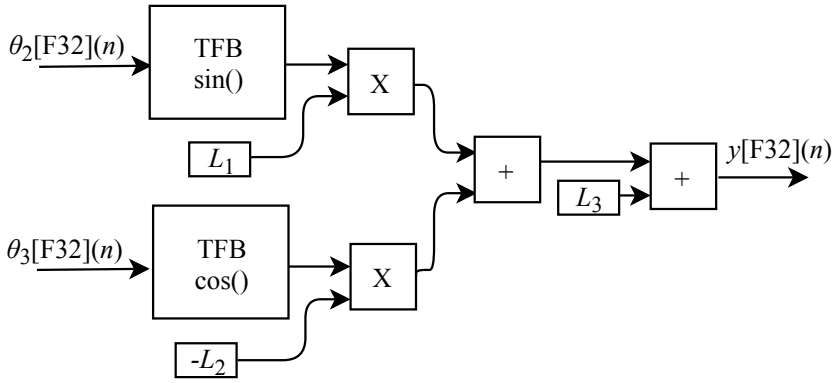


Figure 2.7: Proposed forward kinematics circuit for obtaining the $y[F32](n)$ spatial coordinate (Eq. 2.16) - FK-HMD & FK-HSD.

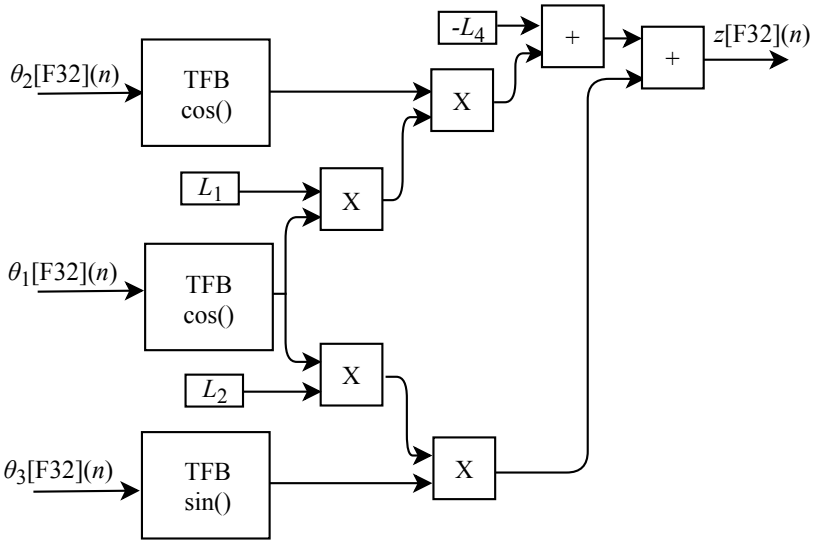


Figure 2.8: Proposed forward kinematics circuit for obtaining the $z[F32](n)$ spatial coordinate (Eq. 2.17) - FK-HMD & FK-HSD.

$\theta_2^{SD}[F32](n)$ and $\theta_3^{SD}[F32](n)$ input signals enter the module via the $\theta^{SD}(n)$ array (eq. 2.49) and after performing all parallel computations, the resulting signals $x^{ENV}(n)$, $y^{ENV}(n)$ and $z^{ENV}(n)$ are output from the module via the $\mathbf{l}(n)$ array (eq. 2.49).

2.6.2 Inverse Kinematics (IK-HSD)

The hardware associated with the slave device (HSD) implements the inverse kinematics through the IK-HSD module, as shown in Figure 2.4. The IK-HSD FPGA-implemented circuit is designed to work with three input signals $x^{HSD}[F32](n)$, $y^{HSD}[F32](n)$ and $z^{HSD}[F32](n)$ and three output signals $\theta_1^{HSD}[F32](n)$, $\theta_2^{HSD}[F32](n)$ and $\theta_3^{HSD}[F32](n)$. However, to calculate $\theta_2^{HSD}[F32](n)$ (eq. 2.24) and $\theta_3^{HSD}[F32](n)$ (eq. 2.25) it is first nec-

essary to perform intermediate calculations to obtain the values of $R[F32](n)$, $r[F32](n)$, $\beta[F32](n)$, $\gamma[F32](n)$ and $\alpha[F32](n)$

Based on Equations 2.18, 2.24 and 2.25, algorithms for calculating $\theta_1^{HSD}[F32](n)$, $\theta_2^{HSD}[F32](n)$ and $\theta_3^{HSD}[F32](n)$ were implemented in FPGA through the generic circuits illustrated in Figures 2.9, 2.10 and 2.11 respectively.

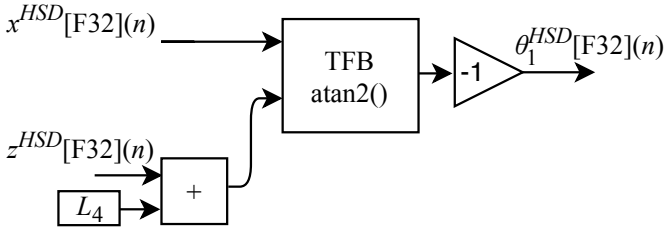


Figure 2.9: Proposed inverse kinematics circuit for obtaining the $\theta_1^{HSD}[F32](n)$ angular position (Eq. 2.18) - IK-HSD.

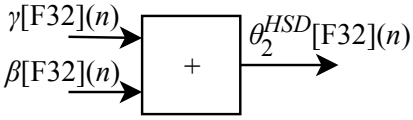


Figure 2.10: Proposed inverse kinematics circuit for obtaining the $\theta_2^{HSD}[F32](n)$ angular position (Eq. 2.24) - IK-HSD.

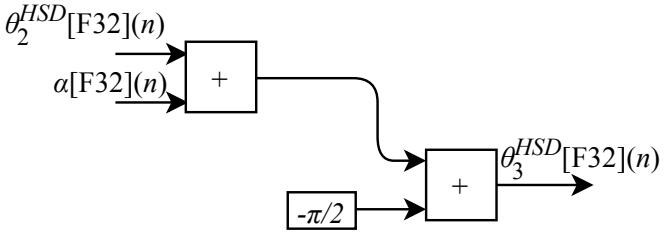


Figure 2.11: Proposed inverse kinematics circuit for obtaining the $\theta_3^{HSD}[F32](n)$ angular position (Eq. 2.25) - IK-HSD.

As already described, and according to the illustrations shown in Figures 2.10 and 2.11, to perform the calculations of $\theta_2^{HSD}[F32](n)$ and $\theta_3^{HSD}[F32](n)$ it is first necessary to perform the intermediate calculations of $\gamma[F32](n)$ (eq. 2.21), $\beta[F32](n)$ (eq. 2.22) and $\alpha[F32](n)$ (eq. 2.23). However, these calculations depend on the calculation of $R[F32](n)$ and $r[F32](n)$. Then, when the IK-HSD module receives the input signals at every n-th instant the circuit shown in Figure 2.9 performs the calculation of $\theta_1^{HSD}[F32](n)$ in parallel with the generic circuits illustrated in Figures 2.12 and 2.13 which were implemented

in FPGA to perform the calculation of $R[F32](n)$ and $r[F32](n)$ based on Equations 2.19 and 2.20.

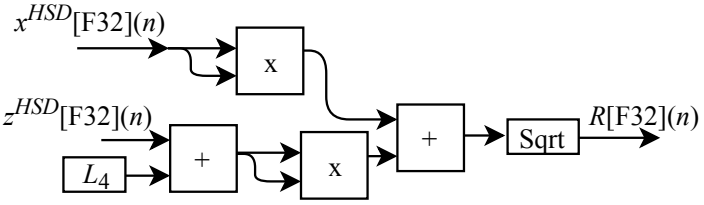


Figure 2.12: Proposed circuit to perform the calculation of $R[F32](n)$ (Eq. 2.19) - IK-HSD.

The circuit shown in Figure 2.12 used to obtain $R[F32](n)$, is designed to work with two input signals $x^{HSD}[F32](n)$ and $z^{HSD}[F32](n)$ and one output signal. This design contains two multipliers, two adders, the L_4 constant and a sub-circuit called *Sqrt*, which was implemented in hardware to calculate the square root.

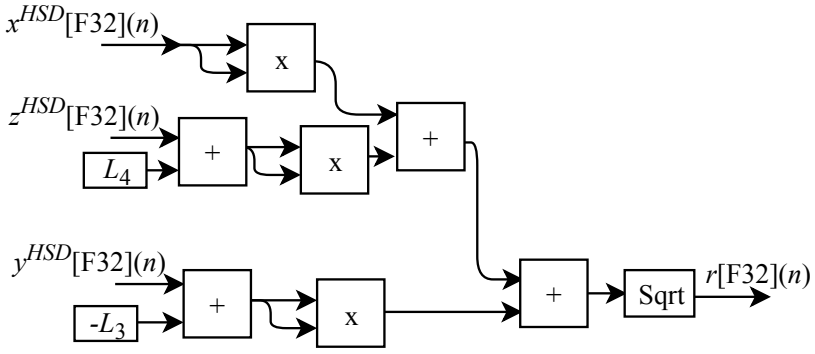


Figure 2.13: Proposed circuit to perform the calculation of $r[F32](n)$ (Eq. 2.20) - IK-HSD.

The $r[F32](n)$ calculation is performed through the circuit shown in Figure 2.13. This circuit is designed to work with three input signals $x^{HSD}[F32](n)$, $y^{HSD}[F32](n)$ and $z^{HSD}[F32](n)$ and one output signal. The circuit consists of three multipliers, four adders, one inverter, the constants L_3 and L_4, and, again, the *Sqrt* sub-circuit.

After the parallel processing of $\theta_1^{HSD}[F32](n)$, $R[F32](n)$ and $r[F32](n)$, the circuits responsible for calculating $\gamma[F32](n)$, $\beta[F32](n)$ and $\alpha[F32](n)$ are also executed in parallel through the FPGA implementations of the generic circuits illustrated in Figures 2.14, 2.15 and 2.16. The value of $\gamma[F32](n)$ is obtained through the circuit shown in Figure 2.14 which is based on the Equation 2.21. The circuit is designed to work with an input signal $r[F32](n)$ and one output signal. It consists of five multipliers, two adder, one divisor, one TFB sub-circuit to calculate the arccosine and the constants L_1 and L_2.

The circuit for obtaining $\beta[F32](n)$ illustrated in Figure 2.15 is based on Equation 2.22 and is designed to work with two input signals $y^{HSD}[F32](n)$ and $R[F32](n)$ and one

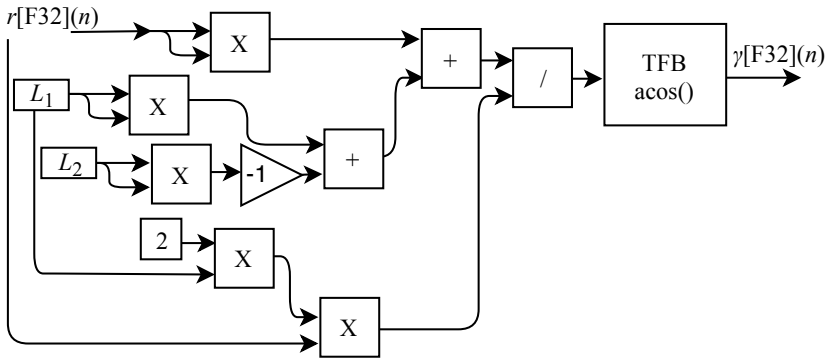


Figure 2.14: Proposed circuit to perform the calculation of $\gamma[F32](n)$ (Eq. 2.21) - IK-HSD.

output signal. The circuit is composed of one adder, one inverter, a TFB sub-circuit to perform the arctangent calculation and the L_3 constant.

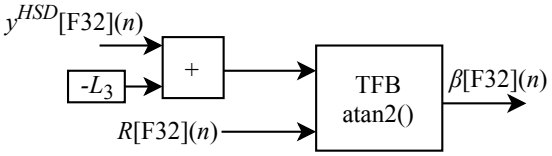


Figure 2.15: Proposed circuit to perform the calculation of $\beta[F32](n)$ (Eq. 2.22) - IK-HSD.

The value of $\alpha[F32](n)$ is obtained from the circuit shown in Figure 2.16 which is based on Equation 2.23 and is designed to work with an input signal $r[F32](n)$ and one output signal. The circuit is composed of five multipliers, two adders, one inverter, one divider, one TFB sub-circuit to perform the arccosine calculation and the constants L_1 and L_2.

To complete the process, after performing the calculations of $\beta[F32](n)$, $\gamma[F32](n)$ and $\alpha[F32](n)$, it is possible to obtain the $\theta_2^{HSD}[F32](n)$ and $\theta_3^{HSD}[F32](n)$ values in parallel through the circuits shown in Figures 2.10 and 2.11.

2.6.3 Kinesthetic Feedback Force (KFF-HMD)

As illustrated in Figure 2.4, the hardware associated with the master device (HMD) implements the kinesthetic feedback force through the KFF-HMD module. Based on Equation 2.26, the KFF-HMD module was implemented in FPGA through the generic circuit illustrated in Figure 2.17. This circuit is composed of sub-circuits that correspond to parts of Equation 2.26. The sub-circuit called JM, described in Equation 2.28, is responsible for calculating the Jacobian matrix. The KFF sub-circuit makes the relationship between the Jacobian matrix (JM) module and the force array from Equation 2.38.

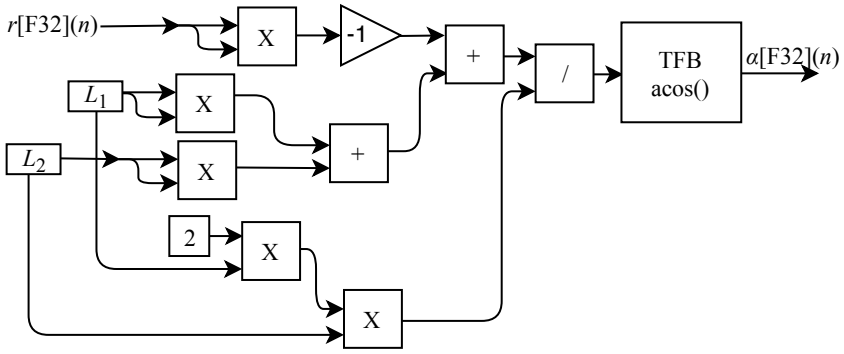


Figure 2.16: Proposed circuit to perform the calculation of $\alpha[F32](n)$ (Eq. 2.23) - IK-HSD.

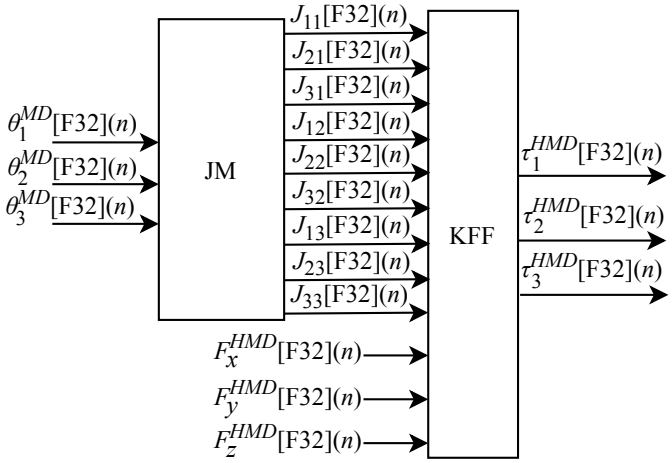


Figure 2.17: Proposed circuit to calculate kinesthetic feedback force(Eq. 2.26) - KFF-HMD.

The circuit shown in Figure 2.17 has the input signals $\theta_1^{MD}[F32](n)$, $\theta_2^{MD}[F32](n)$ and $\theta_3^{MD}[F32](n)$ that are received from the master device (MD) and also the $F_x[F32](n)$, $F_y[F32](n)$ and $F_z[F32](n)$ signals that are received from the hardware associated to the slave device (HSD). The three output signals are: $\tau_1^{HMD}[F32](n)$, $\tau_2^{HMD}[F32](n)$ and $\tau_3^{HMD}[F32](n)$.

The JM module that represents the sub-circuit responsible for performing the Jacobian matrix calculation consists of nine elements: $J_{11}[F32](n)$, $J_{21}[F32](n)$, $J_{31}[F32](n)$, $J_{12}[F32](n)$, $J_{22}[F32](n)$, $J_{32}[F32](n)$, $J_{13}[F32](n)$, $J_{23}[F32](n)$ and $J_{33}[F32](n)$. The calculation of $J_{21}[F32](n)$ based on Equation 2.30 does not have an associated circuit since its value is 0, i.e. $J_{21}[F32](n) = 0$. Based on Equation 2.29, the algorithm for calculating $J_{11}[F32](n)$ was implemented in FPGA according to the generic circuit illustrated in Figure 2.18. The circuit was designed to work with three input signals and one output signal. It uses the constants L_1 and L_2 and has three TFB sub-circuits: two for performing

the cosine calculation and one for obtaining the sine value.

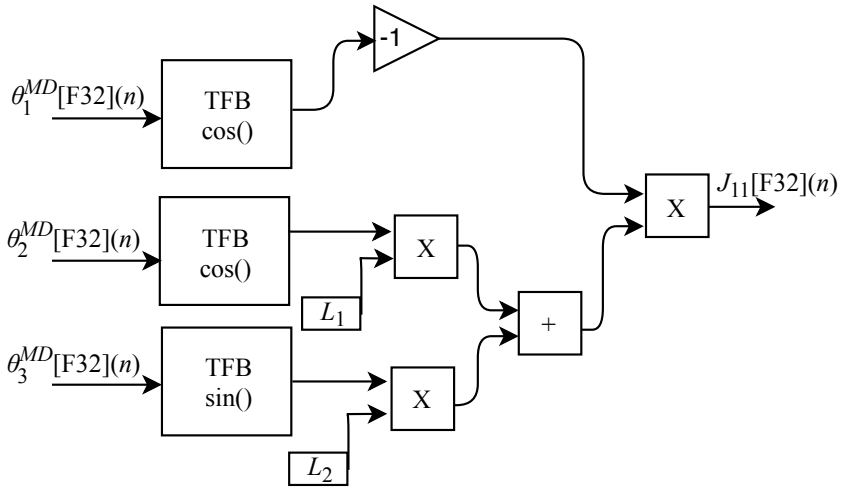


Figure 2.18: Proposed circuit to calculate the Jacobian matrix $J_{11}[F32](n)$ (Eq. 2.29) - JM.

The calculation of $J_{31}[F32](n)$, based on Equation 2.31, was implemented in FPGA according to the generic circuit illustrated in Figure 2.19. The circuit was designed to work with three input signals and one output signal. The circuit has three TFB modules, two for sine calculation and one for cosine value and uses the L_1 and L_2 constants.

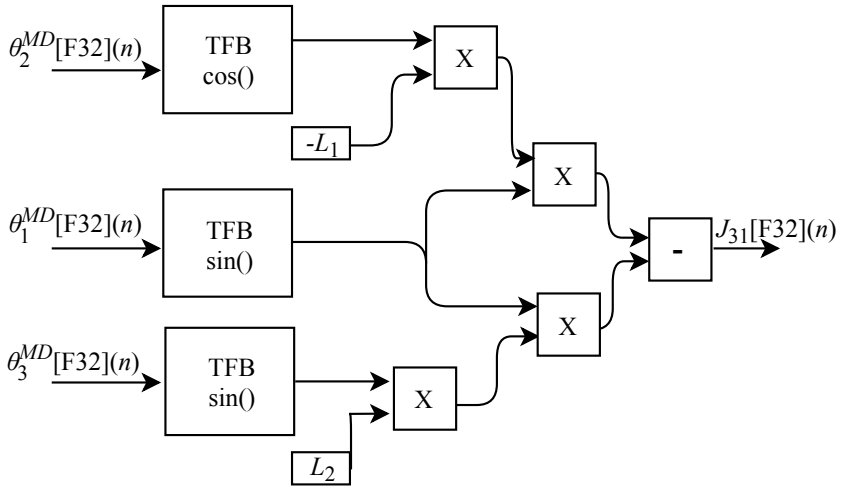


Figure 2.19: Proposed circuit to calculate the Jacobian matrix $J_{31}[F32](n)$ (Eq. 2.31) - JM.

The generic circuit illustrated in Figure 2.20 was implemented in FPGA to perform the calculation of $J_{12}[F32](n)$ and is based on Equation 2.32. The circuit was designed to work with two input signals and one output signal. The circuit has two TFB sub circuits to perform sine calculation and uses the L_1 constant.

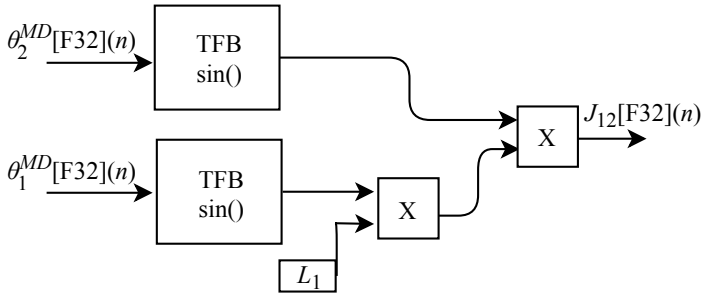


Figure 2.20: Proposed circuit to calculate the Jacobian matrix $J_{12}[F32](n)$ (Eq. 2.32) - JM.

Based on Equation 2.33, the algorithm for calculating $J_{22}[F32](n)$ was implemented in FPGA according to the generic circuit illustrated in Figure 2.21. The circuit was designed to work with one input signal and one output signal. The circuit has a TFB sub-circuit to perform cosine calculation and uses the constant L_1.

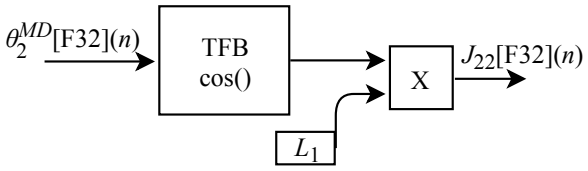


Figure 2.21: Proposed circuit to calculate the Jacobian matrix $J_{22}[F32](n)$ (Eq. 2.33) - JM.

The calculation of $J_{32}[F32](n)$ based on Equation 2.34 was implemented in FPGA according to the generic circuit illustrated in Figure 2.22. The circuit was designed to work with two input signals and one output signal. In addition to the use of the constant L_1, the circuit has two TFB sub circuits, one for performing the cosine calculation and one for the sine.

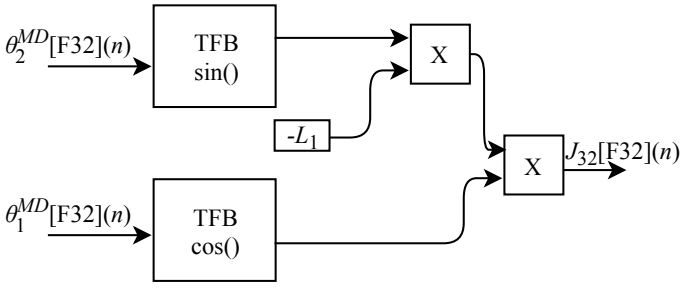


Figure 2.22: Proposed circuit to calculate the Jacobian matrix $J_{32}[F32](n)$ (Eq. 2.34) - JM.

The generic circuit illustrated in Figure 2.23 was implemented in FPGA to perform the

calculation of $J_{13}[F32](n)$ and which is based on Equation 2.35. The circuit was designed to work with two inputs and one output signal. In addition to using the constant L_2, the circuit has two TFB sub circuits, one for performing cosine calculation and one for the sine.

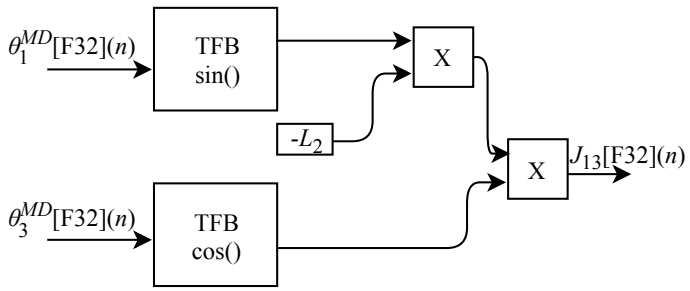


Figure 2.23: Proposed circuit to calculate the Jacobian matrix $J_{13}[F32](n)$ (Eq. 2.35) - JM.

Based on Equation 2.36, the algorithm for calculating $J_{23}[F32](n)$ was implemented in FPGA according to the generic circuit illustrated in Figure 2.24. The circuit was designed to work with one input signal and one output signal. The circuit contains a TFB sub-circuit to perform the sine calculation and uses the L_2 constant.

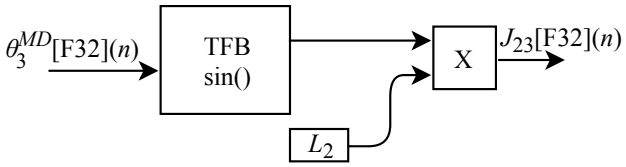


Figure 2.24: Proposed circuit to calculate the Jacobian matrix $J_{23}[F32](n)$ (Eq. 2.36) - JM.

The calculation of $J_{33}[F32](n)$, based on Equation 2.37, was implemented in FPGA according to the generic circuit illustrated in Figure 2.25. The circuit was designed to work with two input signals and one output signal. In addition to the use of constant L_2, the circuit has two TFB sub-circuits to perform the cosine calculation.

All displayed circuits related to the JM sub-circuits are calculated in parallel at each n-th instant. The results are then sent to the KFF module which also performs the calculations of $\tau_1^{HMD}[F32](n)$, $\tau_2^{HMD}[F32](n)$ and $\tau_3^{HMD}[F32](n)$ in parallel. The KF circuit shown in Figure 2.17 is designed to work with twelve input signals and three output signals.

Based on Equation 2.39, the algorithm for calculating $\tau_1^{HMD}[F32](n)$ was implemented in FPGA according to the generic circuit illustrated in Figure 2.26. The circuit was designed to work with six inputs and one output.

The calculation of $\tau_2^{HMD}[F32](n)$ based on Equation 2.40 was implemented in FPGA according to the generic circuit illustrated in 2.27. The circuit was designed to work with six inputs and one output.

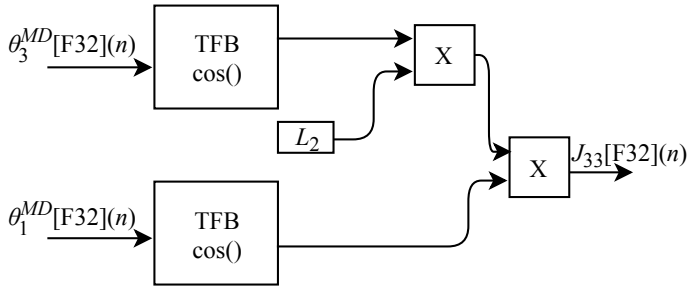


Figure 2.25: Proposed circuit to calculate the Jacobian matrix $J_{33}[F32](n)$ (Eq. 2.37) - JM.

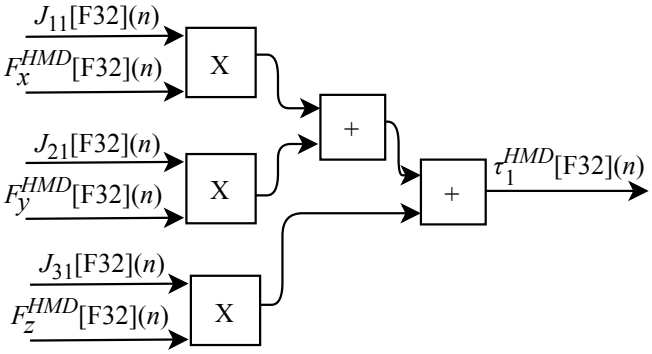


Figure 2.26: Proposed circuit to calculate the torque of the $\tau_1^{HMD}[F32](n)$ joint (Eq. 2.39) - KFF.

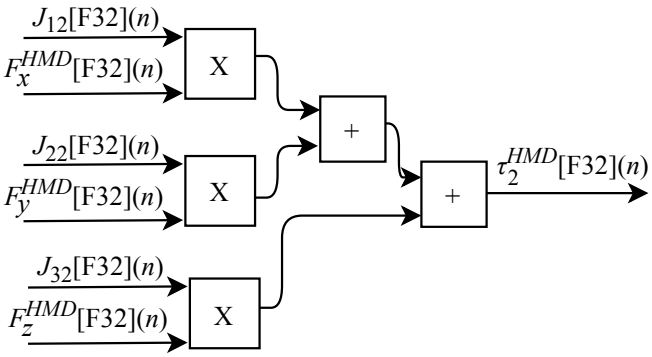


Figure 2.27: Proposed circuit to calculate the torque of the $\tau_2^{HMD}[F32](n)$ joint (Eq. 2.40) - KFF.

The generic circuit illustrated in Figure 2.28 has been implemented in FPGA to perform the calculation of $\tau_3^{HMD}[F32](n)$ and it is based on Equation 2.41. The circuit was designed to work with six inputs and one output.

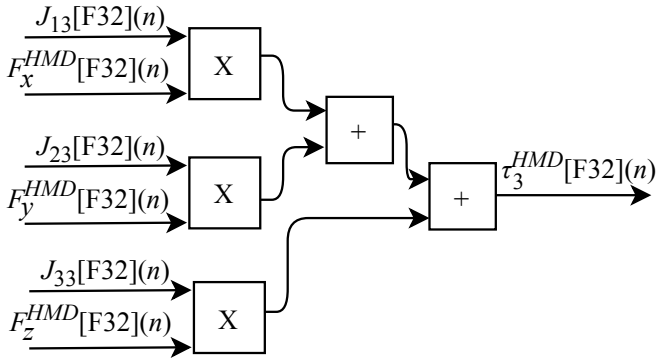


Figure 2.28: Proposed circuit to calculate the torque of the $\tau_3^{HMD}[F32](n)$ joint (Eq. 2.41) - KFF.

2.6.4 Feedback Force (FBF-HSD)

As illustrated in Figure 2.4 the hardware associated with the slave device (HSD) implements the feedback force via the FBF-HSD module. The FPGA-implemented circuit of the FBF-HSD module is designed to work with six input signals and three output signals. Among the six input variables, $x^{OBJ}[F32](n)$, $y^{OBJ}[F32](n)$ and $z^{OBJ}[F32](n)$ represent the spatial position of the closest object to the SD tool and the other three $x^{ENV}[F32](n)$, $y^{ENV}[F32](n)$ and $z^{ENV}[F32](n)$ represent the spatial position of the SD tool in the ENV module. The three outputs $F_x^{HSD}[F32](n)$, $F_y^{HSD}[F32](n)$ and $F_z^{HSD}[F32](n)$ represent the touch of the tool on the object. The variables h_x, h_y and h_z represent the elasticity coefficients associated with the object. All FBF-HSD module calculations are performed in parallel.

Based on Equation 2.60, the algorithm used for calculating $F_x^{HSD}[F32](n)$ was implemented in FPGA according to the generic circuit illustrated in Figure 2.29. The circuit was designed to work with two inputs signals $x^{OBJ}[F32](n)$ and $x^{ENV}[F32](n)$ and one variable h_x.

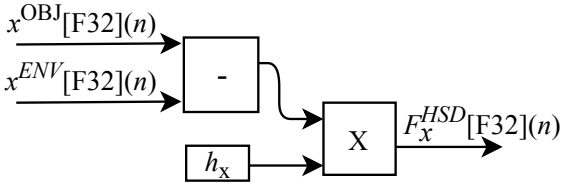


Figure 2.29: Proposed circuit to calculate the feedback force $F_x^{HSD}[F32](n)$ (Eq. 2.60) - FBF-HSD.

The calculation of $F_y^{HSD}[F32](n)$, based on Equation 2.61, was implemented in FPGA according to the generic circuit illustrated in Figure 2.30. The circuit was designed to work with two input signals $y^{OBJ}[F32](n)$ and $y^{ENV}[F32](n)$ and one variable h_y.

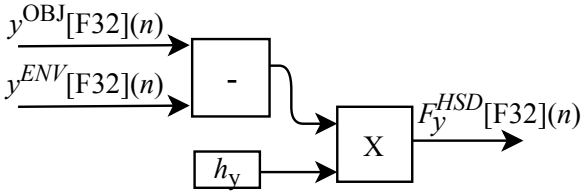


Figure 2.30: Proposed circuit to calculate the feedback force $F_y^{HSD}[F32](n)$ (Eq. 2.61) - FBF-HSD.

The generic circuit shown in Figure 2.31 was implemented in FPGA to perform the calculation of $F_z^{HSD}[F32](n)$ and it is based on Equation 2.62. The circuit was designed to work with two input signals $z^{OBJ}[F32](n)$ and $z^{ENV}[F32](n)$ and one variable h_z.

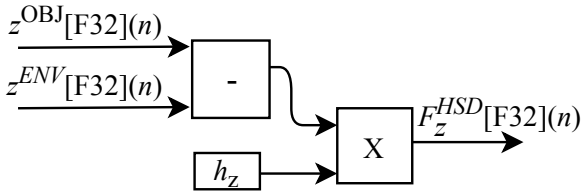


Figure 2.31: Proposed circuit to calculate the feedback force $F_z^{HSD}[F32](n)$ (Eq. 2.62) - FBF-HSD.

2.7 Results Obtained for Synthesis of the Hardware Models in the FPGA

The entire tactile internet model infrastructure presented in Figure 2.4 was implemented with the purpose of validating the FPGA hardware implementation. The results were obtained using the Xilinx Virtex 6 XC6VLX240T-1FF1156 target FPGA. The used Virtex 6 FPGA has 37,680 slices that group 301,440 registers (flip-flops), 150,720 lookup tables (LUTs) that can be used to implement logical functions or memories and 768 DSP cells with multipliers and accumulators.

In order to validate and test the accuracy of the proposed implementation in FPGA analogous hardware implementation a software equivalent of each algorithm described in Section 2.4 was used to compare the software results with the results obtained in the analogous hardware implementation. All modules presented in 2.6 was validated and tested. In all scenarios, the signal sampling rate (or throughput) was $R_s = \frac{1}{t_s}$ (samples per second), where t_s is the period between the n-th samples.

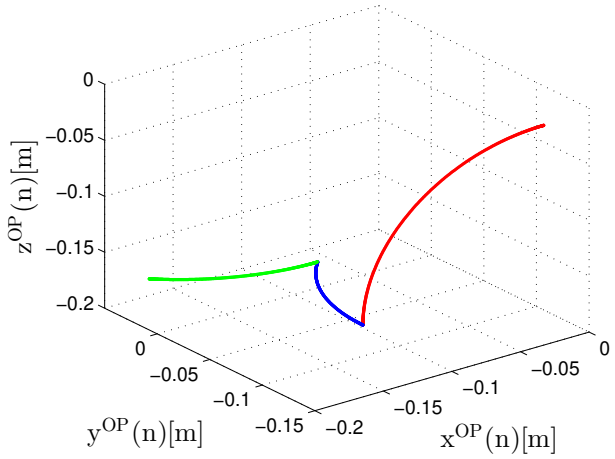


Figure 2.32: Trajectory used to validate hardware modules.

2.7.1 Results and Discussion of Floating-Point Hardware Implementation

A spatial trajectory that represents the data sent by the OP through the $\mathbf{a}(n)$ (Eq. 2.42) signal was created to validate the entire developed environment. The created trajectory performs a variation in all of the three angles of the MD articulation. (Figure 2.3). For this, it was first considered that the MD is in the initial angular position expressed as $\theta_1^{MD}(0) = 0$, $\theta_2^{MD}(0) = 0$ and $\theta_3^{MD}(0) = 0$, which corresponds to the spatial position $x^{OP}(0) = 0$, $y^{OP}(0) = -0.107$ and $z^{OP}(0) = -0.035$ of the tool as illustrated in Figure 2.32. Initially, the first joint is moved to $\theta_1^{MD}(vn) = pi/2$ where v represents a quantity of samples that is equal to 4 seconds, thus resulting in the position $x^{OP}(vn) = -0.132$, $y^{OP}(vn) = -0.107$ and $z^{OP}(vn) = -0.167$. Then, the second joint is moved to $\theta_2^{MD}(vn) = pi/4$ which results in the position $x^{OP}(vn) = -0.093$, $y^{OP}(vn) = -0.013$ and $z^{OP}(vn) = -0.167$ and, finally, the third joint moves up to $\theta_3^{MD}(vn) = pi/4$, thus resulting in the $x^{OP}(vn) = -0.186$, $y^{OP}(vn) = 0.025$ and $z^{OP}(vn) = -0.167$ position. The path created is within the limits of the device workspace and takes a total time of $t_1 = 12$ seconds of which 4 seconds are used to perform the movement of each joint.

In an effort to validate the circuits from the implemented modules in FPGA, equivalent software models were used to compare the results of both implementations. Software models uses a 32-bit floating-point format while the hardware modules run a parallel implementation with a hybrid representation which uses both a 32-bit floating-point and a fixed-point representation in different parts of the proposed architecture, as presented in section 2.6. In all scenarios, the signal sampling rate (or throughput) was $R_s = \frac{1}{t_s}$ (sample per second), where t_s is the time between the n-th samples.

From the experimental results, the mean square error (MSE) between the software model and the hardware implementation proposed by this work was calculated using the

MSE which can be expressed as

$$MSE = \frac{1}{Q}\sum_{n=0}^{Q-1}(M^{SW}[F32](n) - M[F32](n))^2 \tag{2.72}$$

where Q represents the number of tested samples, $M^{SW}[F32](n)$ corresponds to the variables of the software model and $M[F32](n)$ corresponds to the variables of the model implemented in FPGA.

The quantity of tested samples for the results here presented is $Q = 1200$, which correspond to the quantity of samples of the generated trajectory. The variables that correspond to the hardware model $M[F32](n)$ vary according to the module in which it was implemented. In the case of forward kinematics, as the FK-HMD and FK-HSD modules have the same implementation, the values corresponding to the variables $x[F32](n)$, $y[F32](n)$ and $z[F32](n)$ change according to the respective module. For the FK-HMD module, these variables correspond to $x^{HMD}[F32](n)$, $y^{HMD}[F32](n)$ and $z^{HMD}[F32](n)$ and for the FK-HSD module the same variables correspond to $x^{ENV}[F32](n)$, $y^{ENV}[F32](n)$ and $z^{ENV}[F32](n)$ as presented in Section 2.6. For inverse kinematics, the variables $M[F32](n)$ of the IK-HSD module correspond to $\theta_1^{HSD}[F32](n)$, $\theta_2^{HSD}[F32](n)$ and $\theta_3^{HSD}[F32](n)$. For kinesthetic feedback force, the variables $M[F32](n)$ of the KFF-HMD module correspond to $\tau_1^{HMD}[F32](n)$, $\tau_2^{HMD}[F32](n)$ and $\tau_3^{HMD}[F32](n)$. For feedback force, the variables $M[F32](n)$ of the FBF-HSD module correspond to $F_x^{HSD}[F32](n)$, $F_y^{HSD}[F32](n)$ and $F_z^{HSD}[F32](n)$. And finally, in the MSE equation the $M^{SW}[F32](n)$ corresponds to the same variables as the software-implemented model.

Table 2.1 shows the mean square error between the software models and the hardware ones proposed in this work. The obtained MSE-related results proved to be noteworthy, showing that the forward kinematics (FK-HMD and FK-HSD), inverse kinematics (IK-HSD), kinesthetic feedback force (KFF-HMD) and feedback force (FBF-HSD) modules had an acceptable response, even when using a hybrid representation, compared to the software model that uses a floating-point representation. It can be observed that for the variables of the FK-HMD and FK-HSD modules the error was in the range of 10^{-08}, for the IK-HSD module the error was of 10^{-06}, for the variables of the KFF-HMD module the error was of 10^{-07} and for the FBF-HSD module the error was in the range of 10^{-16}. These values demonstrate that the FPGA implementations presented an equivalent behavior to the software models.

In a hardware implementation, it is important to analyze some requirements post-synthesis such as available hardware usage and the execution time. In the case of FPGAs, the resources are measured through the use of LUTs, Registers and Digital Signal Processors (DSPs) units, to name a few. After validating the hardware-implemented models, synthesis results were obtained using the implementation designed for the FPGA target.

Table 2.2 presents the post-synthesis results related to hardware occupancy, sampling rate, and throughput for the modules FK-HMD, KFF-HMD, FK-HSD, IK-HSD, and FBF-HSD. The first column shows the name of the module, the next three columns called registers, LUTs and multipliers represent the amounts of resources used in the FPGA. The column register represents the number of flip-flops that were used, followed by the total

Table 2.1: Mean squared error (MSE) results for floating-point implementation.

Module	Variable	MSE
FK-HMD	$x^{HMD}[F32](n)$	2.333×10^{-8}
	$y^{HMD}[F32](n)$	8.316×10^{-9}
	$z^{HMD}[F32](n)$	1.656×10^{-8}
KFF-HMD	$\tau_1^{HMD}[F32](n)$	1.467×10^{-7}
	$\tau_2^{HMD}[F32](n)$	5.207×10^{-9}
	$\tau_3^{HMD}[F32](n)$	3.350×10^{-7}
FK-HSD	$x^{ENV}[F32](n)$	2.333×10^{-8}
	$y^{ENV}[F32](n)$	8.316×10^{-9}
	$z^{ENV}[F32](n)$	1.656×10^{-8}
IK-HSD	$\theta_1^{HSD}[F32](n)$	3.731×10^{-6}
	$\theta_2^{HSD}[F32](n)$	2.847×10^{-6}
	$\theta_3^{HSD}[F32](n)$	2.702×10^{-6}
FBF-HSD	$F_x^{HSD}[F32](n)$	2.437×10^{-16}
	$F_y^{HSD}[F32](n)$	1.731×10^{-16}
	$F_z^{HSD}[F32](n)$	3.360×10^{-16}

percentage used. The column LUTs represents the number of lookup tables (LUTs) that were used, followed by the total percentage used. And the column multipliers represents the number of DPS48 internal multipliers that were used, followed by the total percentage used. The t_s column represents the sampling rate in nanoseconds that was obtained for each hardware module. Finally, the R_s column displays throughput ($R_s = \frac{1}{t_s}$) values in mega-samples per second for the hardware modules.

Table 2.2: Hardware occupancy, sampling rate and throughput results for floating-point format.

Module Name	Registers (Flip-Flops)	LUTs	Multipliers (DSP48)	t_s (ns)	R_s (MSps)
FK-HMD	3041 (1.01%)	8008 (5.31%)	11 (1.43%)	47	21.27
KFF-HMD	3113 (1.03%)	12251 (8.13%)	48 (6.25%)	70	14.28
FK-HSD	3041 (1.01%)	8008 (5.31%)	11 (1.43%)	47	21.27
IK-HSD	3149 (1.04%)	14107 (9.36%)	27 (3.52%)	218	4.58
FBF-HSD	323 (0.11%)	1236 (0.82%)	9 (1.17%)	21	47.61

The synthesis results presented in Table 2.2 show that the resources used for the FK-HMD and FK-HSD modules were the same. This means that each module, individually, used a percentage of 1.01% which is equivalent to 3,041 of the available hardware resources for the registers, was used 5.31% with LUTs, and 1.43% for embedded multipliers DSP48. The IK-HSD module had a hardware percentage consumption of 1.04% for registers, 9.36% for LUTs and 3.52% for multipliers. The KFF-HMD module had a consumption of 1.03%, 8.13% and 6.25% for registers, LUTs and multipliers, respectively.

Finally, the FBF-HSD module used a percentage of 0.11% for registers, 0.82% for LUTs and 1.17% for multipliers.

Based on data presented in Table 2.2, the HMD modules (FK-HMD and KFF-HMD) that is associated with the MD device has consumed 6,154 (2.04%) for register, 20,259 (13.44%) for LUTs and 59 (7.68%) for multipliers. In the case of hardware associated with the SD device, the HSD modules (FK-HSD, IK-HSD and FBF-HSD) had consumed 6,513 (2.16%) for register, 23,351 (15.49%) for LUTs and 47 (6.12%) for multipliers.

The hardware resources consumed by the HMD hardware modules and the HSD hardware modules were very low. Even if all modules are implemented in single hardware, the consumption remains low. The total sum of hardware resources used in the FPGA by all modules (FK-HMD, KFF-HMD, FK-HSD, IK-HSD and FBF-HSD) was: 12,667 (4.20%) for register, 43,610 (28.93%) for LUTs and 106 (13.80%) for multipliers. The low hardware resources consumption demonstrates that the proposed implementations take up little hardware space in the FPGA which allows other separate implementations to be used concomitantly.

Conform shows the Table 2.2, regarding the throughput values, R_s, the results obtained were significant. Values of 21.27 MSps for the FK-HMD and FK-HSD modules, 4.58 MSps for the IK-HSD module, 14.28 MSps for the KFF-HMD module and 47.61 MSps for the FBF-HSD module were achieved. These results enable critical applications that demand strict time constraints, as it is the case with tactile internet applications.

In Table 2.3, it is possible to see the speedup obtained in relation to latency time constraints. The first column presents the latency constraints of 1 ms and 10 ms that are presented in the literature. The second column shows the minimum latency values that are required for the application to function normally. The third column shows the latency related with the hardware implementation here presented.

Table 2.3: Hardware speedup related to the time limits for the 1 ms and 10 ms latency constraints.

Time Restriction	**Latency Limit**	t_{hardware}	**Speedup**
1 ms	37.5 μs	403 ns	93 $\times$
10 ms	375 μs	403 ns	930 $\times$

The 1 ms restriction corresponds to the maximum latency limit of 37.5 μs for acceptable hardware performance. For the 10 ms constraint, the maximum limit is 375 μs. The value t_{hardware} that is presented in Table 2.3 and according to Equation 2.71, corresponds to the sum of the latencies of the five implemented modules (Table 2.2), two modules are associated with the MD device (FK-HMD and KFF-HMD) and three modules are associated with the SD device (FK-HSD, IK-HSD, and FBF-HSD).

Thus, the presented value of 403 ns in Table 2.3 corresponds to the sum of the two modules related to the master component, which has a total of 117 ns of which 47 ns come from the FK-HMD module and 70 ns from the KFF-HMD module together with the sum of the three modules referring to the slave component, which has a total of 286 ns of which

Table 2.4: Comparative table with state of the art works.

Reference	Device	DoF	Data type	FK	IK	KFF	FBF
This work	Virtex 6	3	Floating P.	47 ns	218 ns	70 ns	21 ns
(Sánchez et al. 2010)	Virtex 2	5	Floating P.	1240 ns	-	-	-
(Gac et al. 2012)	Cyclone IV	3	Floating P.	-	143000 ns	-	-
(Wu et al. 2014)	Unknown	6	Fixed P.	3000 ns	4500 ns	-	-
(Wong & Liu 2013)	Cyclone IV	10	Fixed P.	-	440 ns	-	-
(Linh et al. 2015)	Cyclone IV	5	Fixed P.	680 ns	940 ns	-	-
(Jiang et al. 2017)	Artix 7	3	Fixed P.	2000 ns		-	-

47 ns derives from the FK-HSD module, 218 ns from IK-HSD and 21 ns from the FBS-HSD module. So for the 1 ms constraint, the implementation presented a 93 $\times$ speedup relative to the 37.5 μs, and for the 10 ms constraint, the speedup was of 930 $\times$ relatives to the 375 μs limit.

The sample rates resulted from the five modules that were implemented in this work were notably fast. The values obtained contributed to the hardware meeting the time constraints limits required in a tactile internet environment. Hardware latency showed values significantly below the required constraints, as shown in Table 2.3. These values are well below the 30% presented in the literature and due to the fact that the communication medium demands 70% of application latency, this value can be increased as the latency of hardware devices showed to be significantly low. In other words, it can be said that the remaining latency not spent on the hardware devices can be consumed on the network.

It is important to remember that in a more complex tactile internet environment, there are several others more algorithms to be implemented in hardware such as prediction algorithms, dynamic control, AI based techniques, etc. However, as the proposed implementations presented low hardware resource consumption, other necessary modules, as the ones previously mentioned, could be also implemented in the same shared hardware since resources would still be available.

Comparison with the Related Works

Table 2.4 presents comparisons of the results obtained by the proposed implementation of this work with equivalent results found in works from the state of the art. The first column indicates the references of related works. The next two columns show the used FPGA platform and the amount of degrees of freedom of the used device. The fourth column presents the type of numerical representation used in the implementation and, finally, the last four columns present the times obtained by each reference for latency added by the forward kinematics (FK), inverse kinematics (IK), the kinesthetic force feedback (KFF) and feedback force (FBF) modules, respectively.

As described in Table 2.4, a hardware model for calculating the forward kinematics of a 5-DoF device is presented in (Sánchez et al. 2010). Regarding trigonometric calculations, an expansion of the Taylor series was implemented in FPGA for computing the sine, cosine, and tangent arc functions. The proposed hardware was implemented using a 32-bit floating-point representation. The total time to perform the calculations was

1240 ns. The calculations are performed in parallel. Comparing to the forward kinematics (FK) implementation using 32-bit floating-point proposed by this work, the speedup was 26.38 × over the model presented in (Sánchez et al. 2010).

The work presented in (Gac et al. 2012) shows the results of an implementation of the inverse kinematics module using floating-point 32-bit representation. Three types of implementations are presented, but only the one with the best performance was compared, for that was used as an FPGA and use a microprocessor system based on the Nios II processor. With it, it is possible to perform operations such as multiplication, subtraction, division, sum and square root calculation in hardware was used. The equations allow partial parallelization of individual operations, decreasing the computation time. The kinematic model was designed to work with a 3-DoF device, and the time required to calculate is 143000 ns. When compared with the proposal of inverse kinematics (IK) presented in this work, which uses 32-bit floating-point representation, this implementation presented a speedup of 655.96 × over in relation to the model proposed by (Gac et al. 2012).

The kinematics models presented in (Wu et al. 2014) described in Table 2.4, presented data regarding the forward and inverse kinematics implementations for controlling a 6-DoF device using the 32-bit fixed-point representation. The modules were implemented using 21-bit for the fractional part and 11-bit in the integer part. For the forward kinematics (FK), 3000 ns are required to perform all calculations, and for inverse kinematics (IK), 4500 ns is required. Based on the results of the implementations presented in this section, the implementation proposed for this work using floating-point representation had a speedup of 63.82 × for forward kinematics and 20.64 × for the inverse kinematics.

The research presented in (Wong & Liu 2013) proposed a hardware implementation of inverse kinematics to control a 10-DoF device. Although the robotic model has a quantity of 10 Dof, the equations for the calculations are very simple and similar, the majority being composed of subtraction and division calculations. Regarding trigonometric calculations, only the tangent arc is used in the equations, is it, and the square root used through the CORDIC module. The hardware was projected using the 32-bit fixed-point representation, however the amount of bits used in the fractional part was not specified. The architecture proposed to calculate the inverse kinematics requires 440 ns to perform the computation. All calculations are performed by the hardware in parallel. Comparing to the inverse kinematics (IK) implementation using 32-bit floating-point proposed by this work, the speedup was 2.01 × over the model presented in (Wong & Liu 2013). The processing time has a lower value when considering the DoFs, but this is due to the fact that the algorithms are less complex.

The authors in (Linh et al. 2015) present the results of fixed-point implementation for forward and inverse kinematics to control a 5-DoF device, as described in Table 2.4. The proposed hardware implementation uses the numerical representation of 32-bit (15-bit to fractional part) and 16-bit (7-bit to fractional part) in different parts of the modules. The equations associated with the calculation of the direct and inverse kinematics make use of the trigonometric functions sine, cosine, arctangent, and arccosine. To perform the arctangent and arccosine, the Taylor series extension was used. The time required to perform the calculations is 680 ns and 940 ns for forward and inverse kinematics, respectively. Comparing to the floating-point implementation proposed by this work, the speedup was

14.46 × for forward kinematic and 4.31 × for inverse kinematic over the model presented in (Linh et al. 2015).

Differently from previous works (Table 2.4), in (Jiang et al. 2017), the authors present unique hardware for calculating forward and inverse kinematics together. The hardware computes all calculations in parallel. The computation of direct and inverse kinematics has a single processing time. An ARM processor was used to make the communication part between the modules, and the FPGA is used to calculate the kinematics model. The CORDIC module was used to perform trigonometric calculations. In the proposed model, the 32-bit fixed-point representation was used. The total time to perform the calculation is 2000 ns. The time obtained was calculated taking into account the entire process duration, however, separate times for each module were not specified. Given this scenario, by adding the t_s FK module time that calculates forward kinematics with the IK module, the total time resulting from both implementations reaches 265 ns, according to Table 2.4. Hence, the hardware presented in the work here developed achieved a 7.54 × speedup over the model presented in (Jiang et al. 2017).

It can be seen from Table 2.4, that none of the works from the state-of-the-art presented the hardware implementation of all four robotics algorithms that were presented here. It is also noted that just two works used the floating-point numerical representation. The floating-point implementation of robotics algorithms proposed by this work showed significant gains when compared to the works presented in the literature as shown in Table 2.4. The different amounts of degrees of freedom (DoF) used in the devices can somehow influence in values of sample rate and throughput. Another factor that can also influence these values is in relation to the type of FPGA that is used to perform the synthesis. Due to the fact that the implementation of this work was designed in a parallel architecture, the increase in the amount of DoF does not necessarily reflect in a significant increase in sample rate.

2.7.2 Results and Discussion of Fixed-Point Hardware Implementation

All circuits presented in section 2.6 were implemented in FPGA using a fixed-point numerical representation in three different scenarios. Each scenario was developed using a different quantity of bits for the fractional part. For each implementation, 3 types of signed fixed-point number representation have been used 12-bit, 14-bit, and 16-bit, and all of these implementations were tested and analyzed. For fixed-point implementations, the variables follow a notation expressed as $[sV.N]$ indicating that the variable is formed by V bits of which N bits are dedicated to the fractional part and the s symbol indicates that the variable is signed. In this case, the number of bits dedicated to the integer part is $V - N - 1$.

In the results described in this section, all of the multiplication operations were performed using embedded multipliers of the type DSP48. The operations of addition, subtraction and the CORDIC algorithms associated with the operations of trigonometric functions were implemented with logic cells using fixed-point representation. The operations of division and square root functions were implemented with logic cells using a 32-bit

Table 2.5: Results for the area occupied, sample time and processing speed for different fixed-point formats.

Module Name	**Fixed Point Format** ([*V.N*])	**Registers (Flip-Flops)**	**LUTs**	**Multipliers (DSP48)**	t_s (ns)	R_s (MSps)
FK-HMD	[16.12]	2924 (0.97%)	3182 (2.11%)	4 (0.52%)	10.133	98.68
	[14.10]	2874 (0.95%)	3146 (2.09%)	4 (0.52%)	9.939	100.61
	[12.8]	2843 (0.94%)	3063 (2.03%)	4 (0.52%)	9.119	109.66
KFF-HMD	[16.12]	2925 (0.97%)	3468 (2.30%)	16 (2.08%)	16.008	62.46
	[14.10]	2874 (0.95%)	3409 (2.26%)	16 (2.08%)	15.574	64.20
	[12.8]	2844 (0.94%)	3187 (2.11%)	16 (2.08%)	14.333	69.76
FK-HSD	[16.12]	2924 (0.97%)	3468 (2.11%)	4 (0.52%)	10.133	98.68
	[14.10]	2874 (0.95%)	3146 (2.09%)	4 (0.52%)	9.939	100.61
	[12.8]	2843 (0.94%)	3063 (2.03%)	4 (0.52%)	9.119	109.66
IK-HSD	[16.12]	2959 (0.98%)	8103 (5.38%)	9 (1.17%)	180.811	5.53
	[14.10]	2475 (0.82%)	7543 (5.00%)	9 (1.17%)	179.844	5.56
	[12.8]	1889 (0.63%)	7016 (4.65%)	9 (1.17%)	177.255	5.64
FBF-HSD	[16.12]	144 (0.048%)	64 (0.042%)	3 (0.39%)	8.939	111.86
	[14.10]	126 (0.042%)	59 (0.039%)	3 (0.39%)	8.864	112.81
	[12.8]	108 (0.036%)	45 (0.030%)	3 (0.39%)	8.691	115.06

floating-point arithmetic representation.

Table 2.5 presents the results obtained after the process of synthesis in the FPGA target for the FK-HMD and KFF-HMD modules, which are associated with the master device, and the FK-HSD, IK-HSD, and FBF-HSD modules, which are associated with the slave device. For each module it is possible to see three different fixed-point implementations which differ in the quantity of bits. The number of bits used in the integer and fractional part is indicated in the first column of the table. The next three columns show the occupation area used by the registers (flip-flops), LUTs and embedded multipliers (DSP48) and the last two columns show the sampling rate (t_s) in nanoseconds and throughput (R_s) values in mega-samples per second.

Given the results shown in Table 2.5, it can be noted that the module associated with forward kinematics (FK-HMD or FK-HSD), have consumed less than 1% in resource usage of registers in each the implementations. The hardware resources occupied by LUTs were no greater than 2.11% for the 16-bit implementation. The FPGA resources used by the KFF-HMD module were no more than 1% for registers and the occupation of the LUTs did not exceed a percentage of 2.3% in any of the implementations. For the IK-HSD module, the hardware resources used were no more than 1% for registers as well and was no greater than 5.38% regarding LUTs usage. Finally, the FPGA resources associated with registers and LUTs used by the FBF-HSD module were no more than 0.048% in any implementation.

How the multiplication operations were synthesized in internal DSP circuits, the number of used multipliers remained constant regardless of the number of bits used in each implementation. As can be seen from Table 2.5, the modules associated with the forward kinematics (FK-HMD or FK-HSD) have used $\approx 0.5\%$ of the available DSP48 multipliers in the FPGA. The modules KFF-HMD, IK-HSD and FBF-HSD used $\approx 2\%$, $\approx 1\%$ and $\approx 0.4\%$ respectively. According to the obtained synthesis results, occupation rate of the proposed implementations takes up little space in the FPGA. Regarding the number

Table 2.6: Hardware speedup related to the time limits for the 1 ms and 10 ms latency constraints.

Time Restriction	Latency Limit	**Fixed-Point** [16.12]		**Fixed-Point** [14.10]		**Fixed-Point** [12.8]	
		t_{hardware}	**Speedup**	t_{hardware}	**Speedup**	t_{hardware}	**Speedup**
1 ms	37.5 μs	226 ns	$\approx 165\times$	224 ns	$\approx 167\times$	218 ns	$\approx 172\times$
10 ms	375 μs	226 ns	$\approx 1659\times$	224 ns	$\approx 1674\times$	218 ns	$\approx 1720\times$

of registers used, the amount used in each implementation did not exceed 1%, and the resources associated with LUTs did not exceed 5.38%.

Among the modules that were implemented, the one that consumed the most hardware resources was the IK-HSD, and the one that consumed the least was the FBF-HSD. This is due to the difference in the complexity of the circuits. The IK-HSD module, because it is the solution of a system of non-linear equations, the circuit becomes more complex in relation to the others. The FBF-HSD module, use a simpler mathematical equation, which only uses multiplication and addition, this makes it less complex compared to others.

Regarding the throughput values, R_s, the results presented in Table 2.5 show that fixed-point implementations performed very similarly. The throughput obtained for the forward kinematic modules (FK-HMD or FK-HSD) was between 98.68 MSps and 109.66 MSps for 16-bit and 12-bit implementations, respectively. This value shows that the 12-bit implementation has 10.98 MSps more throughput, that is, it is 1.11× faster than the 16-bit implementation. For the modules, IK-HSD, KFF-HMD, and FBF-HSD the throughput of 12-bit implementation was 1.11×, 1.01× and 1.02× higher than the implementation of 16-bit. The difference between these implementations was of 0.11 MSps, 7.30 MSps, and 3.20 MSps for IK-HSD, KFF-HMD, and FBF-HSD.

In general, the hardware resources occupancy rate and throughput are sensitive to the number of bits used in hardware implementations. In this work, according to the synthesis results that were obtained (Table 2.5), the proposed fixed-point implementations take up little resources in the FPGA and the throughput values were very close for the implementations of 16-bit, 14-bit, and 12-bit. Typically, by using a higher bit representation an increase in resources consumption is noted in terms of area compared to implementation with fewer bits. On the other hand, when the implementation has a smaller amount of bits the throughput tends to increase. As a consequence, the higher the number of bits allocated to the fractional part, the higher the accuracy. Hence, depending on the required accuracy level for a given hardware module, the relationship between throughput vs accuracy has to be carefully taken into consideration.

In Table 2.6, it is possible to see the speedup obtained by the fixed-point implementations in relation to the latency constraints of 1 ms and 10 ms that are presented in Section 2.5. As previously described, the maximum latency limit for the 1 ms and 10 ms are 37.5 μs and 375 μs, respectively. The value t_{hardware} that is presented in Table 2.6, corresponds to the sum of the latencies (t_s) of the five implemented modules (Table 2.5), of which two modules are associated with the MD device (FK-HMD and KFF-HMD) and three modules are associated with the SD device (FK-HSD, IK-HSD, and FBF-HSD).

For the 16-bit implementation, the value of $t_{hardware}$ corresponds to 226 ns, as pre-

sented in Table 2.6. Part of this value corresponds to the sum of the delay (t_s), associated with the two modules related to the MD hardware which has a total of 26.141 ns, with the sum of the delay (t_s) associated with the three modules referring to the SD hardware, which has a total of 199.883 ns. Hence, for the 1 ms constraint, the 16-bit implementation presented a $\approx 165\times$ speedup relative to the 37.5 μs, and for the 10 ms constraint, the speedup was of $\approx 1659\times$ relatives to the 375 μs limit. The 14-bit implementation presented a $\approx 167\times$ speedup relative to the 1 ms constraint and presented a $\approx 1674\times$ speedup relative to the 10 ms constraint. And finally,the 12-bit implementation presented a $\approx 172\times$ speedup relative to the 1 ms constraint and presented a $\approx 1720\times$ speedup relative to the 10 ms constraint.

After performing the synthesis process of all modules (FK-HSD, FK-HMD, IK-HSD, KFF-HMD, and FBF-HSD), a validation process and analysis of the accuracy of the data resulting from the calculations for the different implementations (16-bit, 14-bit, and 12-bit) were performed. For this, the equivalent software models were used to compare the results with the proposed fixed-point hardware.

In order to validate and obtain the squared error between the software implementation and the fixed-point implementations, was used one specific joint position of the trajectory presented in Figure 2.32. The joint position $\theta_1^{MD} = 0.112$, $\theta_2^{MD} = 0.312$ and $\theta_3^{MD} = 1.212$, that represents the three angles of the MD articulation, and that corresponds to the spatial positioning of the tool $x^{OP} = -0.028$, $y^{OP} = 0.19$ and $z^{OP} = 0.080$ were used to calculate the squared error.

For each scenario 100 samples were used to generate the squared error (SE) between the software model and the hardware implementation proposed and that was calculated using the equation which can be expressed as

$$SE = (M^{SW}[F32](n) - M[V.N](n))^2 \tag{2.73}$$

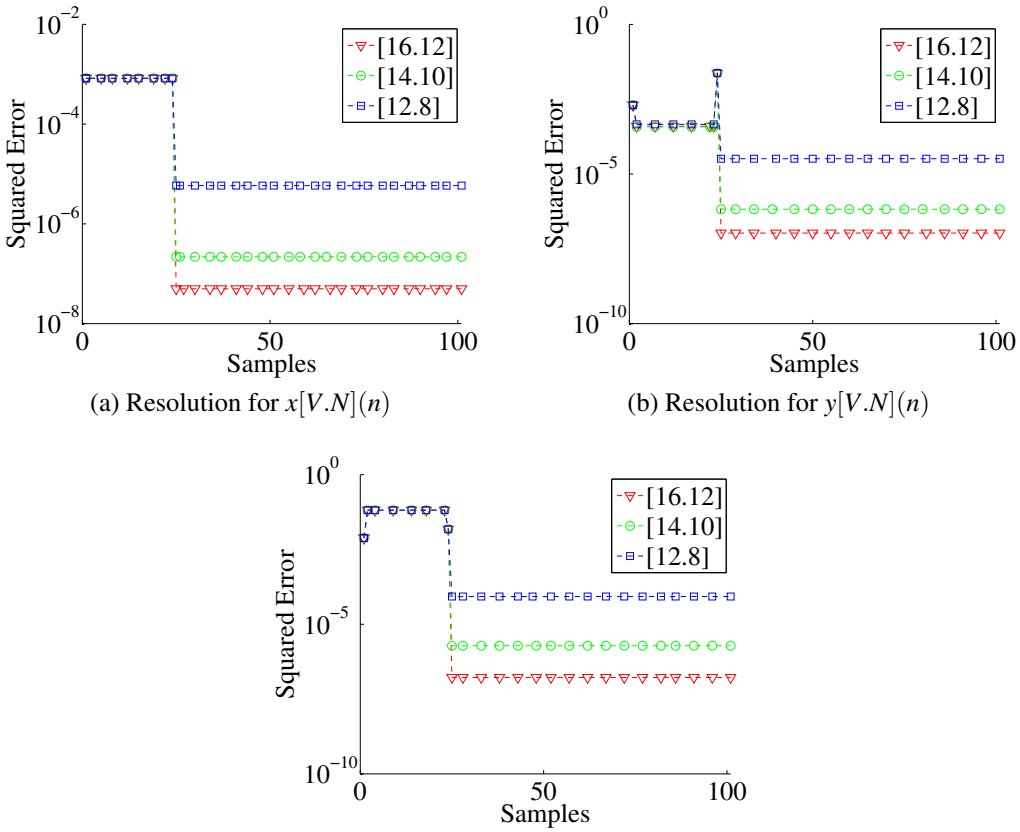
where $M^{SW}[F32](n)$ corresponds to the variables of the software model and $M[V.N](n)$ corresponds to the variables of the model implemented in FPGA.

Table 2.7 shows the results obtained from the squared error for each module and, as can be seen, the higher the resolution used, the lower the squared error.

For a simplified visualization of the difference between implementations resolutions for each variable, Figures 2.33, 2.34, 2.35 and 2.36 were created based on the data presented in Table 2.7. Figure 2.33 shows the squared error for the FK-HMD and FK-HSD modules. For FK-HMD, the variables $x(n)$, $y(n)$ and $z(n)$ presented in Figure 2.33 represents $x^{HMD}[V.N](n)$, $y^{HMD}[V.N](n)$ and $z^{HMD}[V.N](n)$ variables for fixed-point implementations as shown in Table 2.7. These values correspond to the squared error in relation to the software implementation, uses $[F32]$ floating-point variables. For the FK-HSD module the variables presented in Figure 2.33 represents $x^{ENV}[V.N](n)$, $y^{ENV}[V.N](n)$ and $z^{ENV}[V.N](n)$ as shown in Table 2.7.

Figure 2.34 shows the values corresponding to the squared error for the IK-HSD module in relation to the software implementation, which also uses $[F32]$ floating-point variables. The variables presented in Figure 2.34 represent $\theta_1^{HSD}[V.N](n)$, $\theta_2^{HSD}[V.N](n)$ and $\theta_3^{HSD}[V.N](n)$ as shown in Table 2.7.

Figure 2.35 shows the values corresponding to the squared error for the KFF-HMD

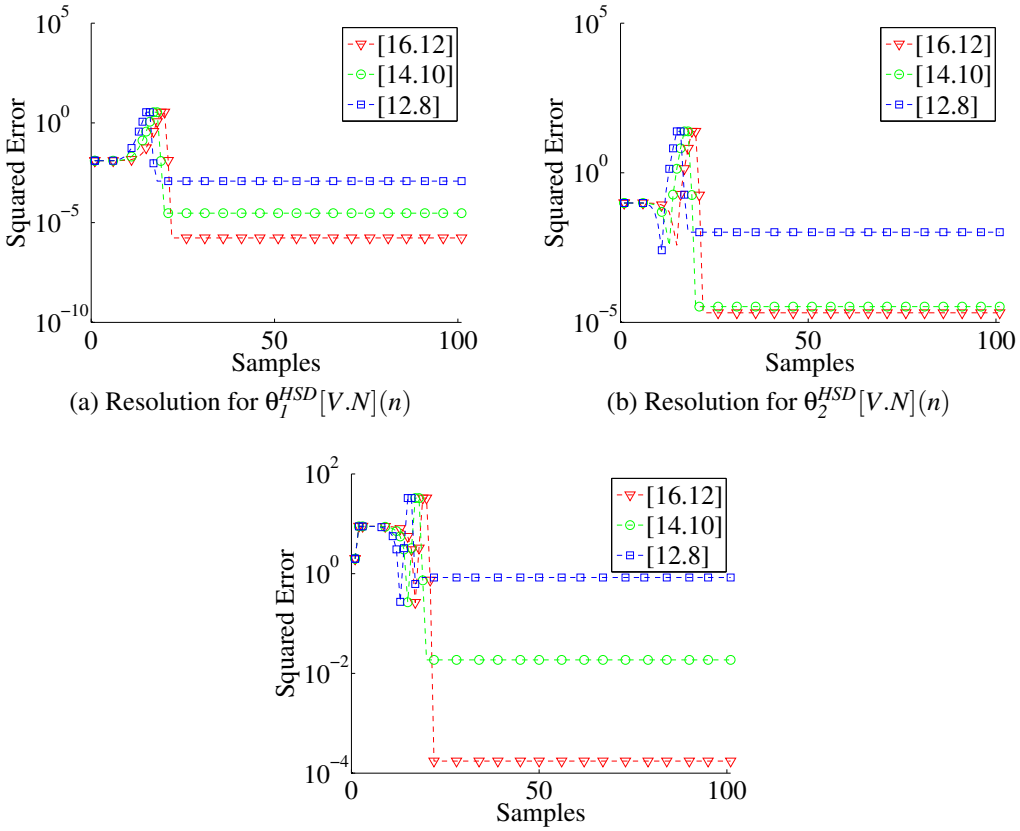


(a) Resolution for $x[V.N](n)$

(b) Resolution for $y[V.N](n)$

(c) Resolution for $z[V.N](n)$

Figure 2.33: The squared error of the FK-HMD and FK-HSD in a test for different fixed-point formats.



(a) Resolution for $\theta_1^{HSD}[V.N](n)$

(b) Resolution for $\theta_2^{HSD}[V.N](n)$

(c) Resolution for $\theta_3^{HSD}[V.N](n)$

Figure 2.34: The squared error of the IK-HMD in a test for different fixed-point formats.

Table 2.7: Squared error results for different fixed-point formats.

Module	Variable	Squared Error (*SE*)		
		Fixed-Point [16.12]	**Fixed-Point** [14.10]	**Fixed-Point** [12.8]
FK-HMD	$x^{HMD}[V.N](n)$	4.985×10^{-8}	2.184×10^{-7}	5.859×10^{-6}
	$y^{HMD}[V.N](n)$	1.060×10^{-7}	6.623×10^{-7}	3.245×10^{-5}
	$z^{HMD}[V.N](n)$	1.658×10^{-7}	1.915×10^{-6}	8.457×10^{-5}
KFF-HMD	$\tau_1^{HMD}[V.N](n)$	2.286×10^{-6}	1.375×10^{-5}	7.382×10^{-5}
	$\tau_2^{HMD}[V.N](n)$	1.174×10^{-6}	1.812×10^{-5}	1.456×10^{-4}
	$\tau_3^{HMD}[V.N](n)$	4.418×10^{-7}	8.191×10^{-6}	1.139×10^{-4}
FK-HSD	$x^{ENV}[V.N](n)$	4.985×10^{-8}	2.184×10^{-7}	5.859×10^{-6}
	$y^{ENV}[V.N](n)$	1.060×10^{-7}	6.623×10^{-7}	3.245×10^{-5}
	$z^{ENV}[V.N](n)$	1.658×10^{-7}	1.915×10^{-6}	8.457×10^{-5}
IK-HSD	$\theta_1^{HSD}[V.N](n)$	1.701×10^{-6}	2.975×10^{-5}	1.200×10^{-3}
	$\theta_2^{HSD}[V.N](n)$	2.089×10^{-5}	3.354×10^{-5}	1.030×10^{-2}
	$\theta_3^{HSD}[V.N](n)$	1.746×10^{-4}	1.860×10^{-2}	8.387×10^{-1}
FBF-HSD	$F_x^{HSD}[V.N](n)$	2.757×10^{-8}	4.281×10^{-7}	2.659×10^{-6}
	$F_y^{HSD}[V.N](n)$	4.206×10^{-8}	4.206×10^{-8}	1.396×10^{-6}
	$F_z^{HSD}[V.N](n)$	2.113×10^{-8}	1.258×10^{-6}	9.455×10^{-6}

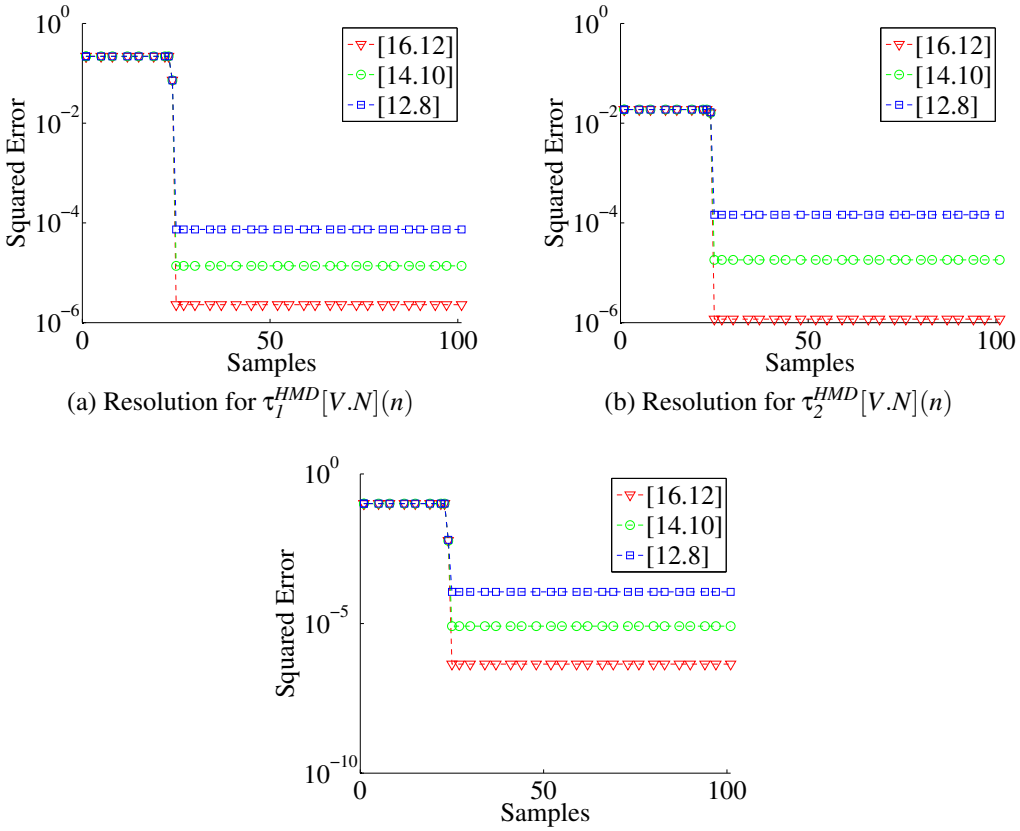
module through the variables $\tau_1^{HMD}[V.N](n)$, $\tau_2^{HMD}[V.N](n)$ and $\tau_3^{HMD}[V.N](n)$ as shown in Table 2.7.

Figure 2.36 shows the values corresponding to the squared error for the FBF-HSD module through the variables $F_x^{HMD}[V.N](n)$, $F_y^{HMD}[V.N](n)$ and $F_z^{HMD}[V.N](n)$ as shown in Table 2.7.

The results obtained through the fixed-point implementations showed to be equivalent to the results obtained by the software implementation that uses floating-point representation, although slight differences regarding accuracy were noted. The difference in response accuracy between fixed-point implementations is due to the gaps left between numbers on fractional parts. Whenever a hardware module generates a new number through a mathematical calculation, that number must be rounded to the nearest value that can be stored by the format in use, hence, this rounding of numbers during signal processing naturally produces a quantization error.

As per the presented Figures (2.33, 2.34, 2.35, 2.36), it is possible to notice that the higher the number of bits in each implementation, the higher the data precision. Only the IK-HSD module presented a lower squared error when compared to the other modules that were implemented. Implementations using the format [14.10] and [12.8] achieved the squared error for the $\theta_3^{HSD}[14.10](n)$ and $\theta_3^{HSD}[12.8](n)$ variable of 1.860×10^{-2} and 8.387×10^{-1}, respectively. This is because the rounding performed for the formats was generating an error in the calculations involved in the process.

Given these facts, for the hardware modules proposed in this work, it is more appropriate to use the implementation with the most bits, because besides having a low resource consumption, the accuracy of the results will be superior.

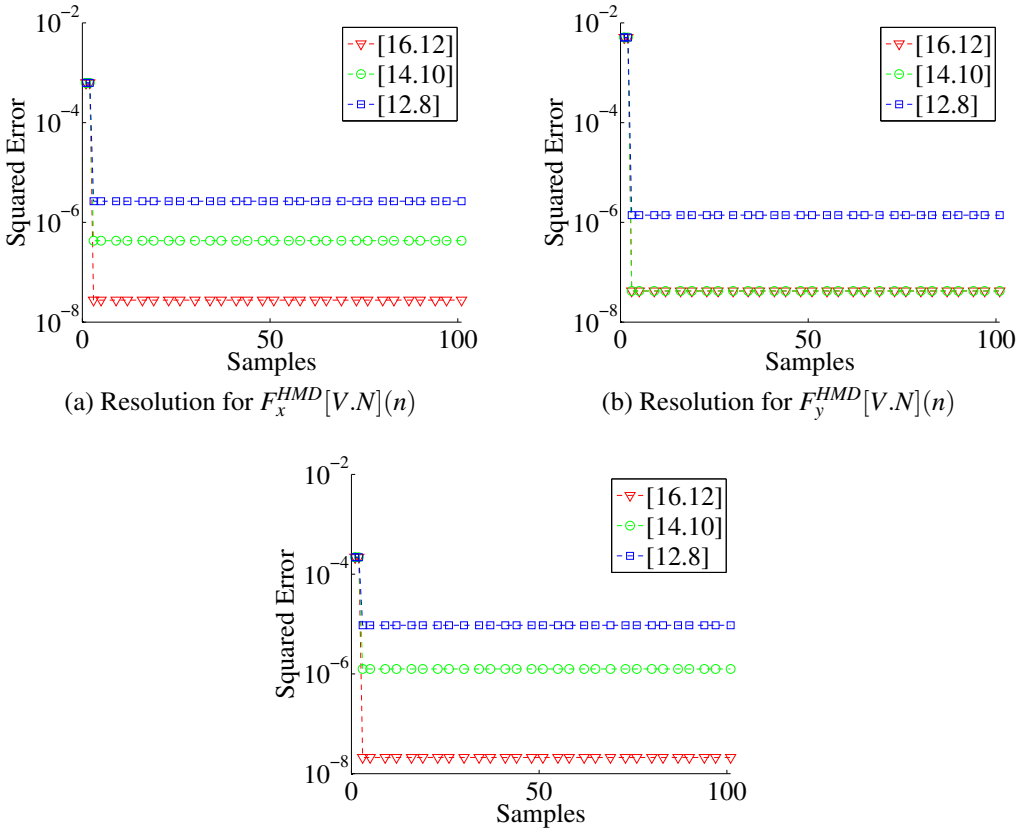


(a) Resolution for $\tau_1^{HMD}[V.N](n)$

(b) Resolution for $\tau_2^{HMD}[V.N](n)$

(c) Resolution for $\tau_3^{HMD}[V.N](n)$

Figure 2.35: The squared error of the KFF-HMD in a test for different fixed-point formats.



(a) Resolution for $F_x^{HMD}[V.N](n)$

(b) Resolution for $F_y^{HMD}[V.N](n)$

(c) Resolution for $F_z^{HMD}[V.N](n)$

Figure 2.36: The squared error of the FBF-HSD in a test for different fixed-point formats.

Comparison with the Related Works

Table 2.8 presents comparisons of the results obtained by the implementation using fixed-point representation proposed by this work with equivalent results found in works from the state of the art already presented in Table 2.4. In Table 2.8 it can be observed that besides the related works, the data related to the post-synthesis results of floating-point implementations presented in Section 2.7.1 is also present. The first column indicates the references of each related work. The next two columns show the used FPGA platform and the amount of degrees of freedom of the used device. The fourth column presents the type of numerical representation used in the implementation. Next, the column called module specifies the FK, IK, KFF, and FBF names that represent the implementations of forward kinematics, inverse kinematics, kinesthetic feedback force, and feedback force, respectively. The column t_s shows the times obtained by each reference for the latency added by each module. The last three columns show the speedup between the fixed-point implementations here presented and the related works.

Table 2.8: Comparative table of the different fixed-point implementations with state of the art works.

Reference	**Device**	**DoF**	**Data type**	**Module**	t_s (ns)	**Speedup** 16-bit	**Speedup** 14-bit	**Speedup** 12-bit
Section 2.7.1	Virtex 6	3	Floating P. 32-bit	FK	47	4.639×	4.728×	5.154×
				IK	218	1.205×	1.212×	1.229×
				KFF	70	4.372×	4.494×	4.883×
				FBF	21	2.349×	2.369×	2.416×
(Sánchez et al. 2010)	Virtex 2	5	Floating P. 32-bit	FK	1240	122.372×	124.761×	135.979×
				IK	-	-	-	-
				KFF	-	-	-	-
				FBF	-	-	-	-
(Gac et al. 2012)	Cyclone IV	3	Floating P. 32-bit	FK	-	-	-	-
				IK	143000	790.881×	795.133×	806.747×
				KFF	-	-	-	-
				FBF	-	-	-	-
(Wu et al. 2014)	Unknown	6	Fixed P.	FK	3000	296.062×	301.841×	328.983×
				IK	4500	24.887×	25.021×	25.387×
				KFF	-	-	-	-
				FBF	-	-	-	-
(Wong & Liu 2013)	Cyclone IV	10	Fixed P.	FK	-	-	-	-
				IK	440	2.433×	2.446×	2.482×
				KFF	-	-	-	-
				FBF	-	-	-	-
(Linh et al. 2015)	Cyclone IV	5	Fixed P.	FK	680	67.107×	68.417×	74.569×
				IK	940	5.198×	5.226×	5.303×
				KFF	-	-	-	-
				FBF	-	-	-	-
(Jiang et al. 2017)	Artix 7	3	Fixed P.	FK	2000	10.474×	10.538×	10.731×
				IK	-	-	-	-
				KFF	-	-	-	-
				FBF				

According to the data presented in Table 2.8, first work, Section 2.7.1, developed the same hardware modules as the ones presented in this work, however, a different numeric representation was used. Based on the delay information (t_s) regarding the implementations of 16-bit, 14-bit, and 12-bit described in Table 2.5, the 16-bit fixed-point implementation had a speedup of 4.639× for forward kinematics (FK-HMD or FK-HSD), 1.205× for the inverse (IK-HSD), 4.372× for kinesthetic feedback force (KFF-HMD) and 2.349× for feedback force (FBF-HSD) over to the implementation using floating-point, as described in Table 2.8. Based on the implementation using a 14-bit fixed-point representation, the speedup of the modules FK, IK, KFF, and FBF were 4.728×, 1.212×, 4.494× and 2.369×, respectively. And for the implementation using a 12-bit fixed-point representation, the speedup was 5.154× for the FK module, 1.229× for the IK module, 4.883× for the KFF module and 2.416× for the FBF module.

Given these results, it is possible to observe that almost all the fixed-point implementations showed significant gains when compared to floating-point implementations. Almost all modules achieved more than 2× processing gains, except for the IK-HSD module which obtained a low speedup of no more than 1.5× regardless of the data representation type used in the implementation. The reason for that is the floating-point sub-circuits, such as division and square root, implemented in the IK-HSD module.

As previously described, the authors in (Sánchez et al. 2010) presented a hardware model for calculating the forward kinematics of a 5-DoF device. The proposed hardware was implemented using a 32-bit floating-point representation and the total time to perform the calculations was of 1240 ns. As shown in Table 2.8, the fixed-point implementation that uses the 16-bit representation proposed in this work presented a speedup of 122.372× more than the proposal presented in (Sánchez et al. 2010). The 14-bit and 12-bit fixed-point implementations presented a 124.761× and 135.979× speedup respectively for the implementation of forward kinematics.

The work presented in (Gac et al. 2012) showed the implementation results of the inverse kinematics module using the 32-bit floating-point representation which resulted in a required calculation time of 143000 ns. The fixed-point implementations using the 16-bit, 14-bit and 12-bit representations for calculating the inverse kinematics (IK) proposed by this work presented a speedup of 790.881×, 795.133×, 806.747×, over the results presented in (Gac et al. 2012), as can be seen from Table 2.8.

In Table 2.8, it is possible to see the work (Wu et al. 2014) that showed the post-synthesis results of two hardware modules responsible for calculating the forward and reverse kinematics of a 6-DoF device. The hardware modules were designed using fixed-point numerical representation. The module responsible for calculating forward kinematics (FK) required 3000 ns to perform all calculations. The inverse kinematics (IK) module took 4500 ns to perform the process. Based on the results of the implementations presented in this section, for the proposed 16-bit fixed-point implementation a speedup of 296.062× for forward kinematics (FK) and 24.887× for the inverse (IK) modules, were, respectively, achieved. The 14-bit implementation demonstrated a speedup of 301.841× for forward kinematics (FK) and 25.021× for the inverse (IK). The 12-bit implementation obtained a speedup of 328.983× and 25.387× for the FK and IK modules, respectively.

The authors in (Wong & Liu 2013) presented a hardware model for calculating the

inverse kinematics of a 10-DoF device. The proposed hardware was implemented using a fixed-point representation and the total time to perform the calculations was of 440 ns. As shown in Table 2.8, the proposed implementation that uses the 16-bit representation in this work presented a speedup of 2.433× over the proposal presented in (Wong & Liu 2013). The 14-bit and 12-bit fixed-point implementations presented a 2.446× and 2.482× speedups, respectively, for the inverse kinematics implementation.

The kinematics models introduced by (Linh et al. 2015) and described in Table 2.8, presented results regarding the forward and inverse kinematics implementations for controlling a 5-DoF device using the 32-bit fixed-point representation. The time required to perform the calculations were of 680 ns and 940 ns for forward and inverse kinematics, respectively. Comparing to the 16-bit implementation proposed by this work, the speedup was of 67.107× and 5.198× for forward and inverse kinematics, respectively. The proposed 14-bit implementation had a speedup of 68.417× for forward kinematics (FK) and 5.226× for the inverse (IK). The 12-bit implementation had a speedup of 74.569× and 5.303× for the FK and IK modules, respectively.

In (Jiang et al. 2017), the authors presented a single hardware implementation for calculating forward and inverse kinematics together. In the model, a 32-bit fixed-point representation was used. The architecture proposed to calculate the kinematics required 2000 ns to perform the computation. Based on the 16-bit implementation proposed for this work, the inverse kinematics module needs 10.133 ns to perform the calculations and the inverse kinematics needs 180.811 ns, as described in 2.5. The two modules together need in total 190.944 ns and that represents a speedup of 10.474× compared to the proposed hardware in (Jiang et al. 2017). For the 14-bit implementation, the two modules together need in total 189.783 ns and that represents a speedup of 10.538×. And for the 12-bit implementation, the two modules together need 186.374 ns and that represents a speedup of 10.731×.

Regarding hardware occupancy, the fixed-point hardware implementations generated a reduction in hardware area consumption, as seen in Table 2.9. The impact of the area reduction obtained becomes even clearer when analyzing the Reduction Rate of Area (RA), that is, how smaller the fixed-point hardware is in relation to the floating-point hardware. The RA value is obtained through the equation

$$RA = \frac{Q_{Before}}{Q_{After}} \tag{2.74}$$

where Q_{Before} is the value corresponding to each hardware resource that was used by the floating-point hardware and Q_{After} is the value corresponding to each hardware resource that was used by the fixed-point hardware. These values of resources can be applied to the number of registers, number of LUTs and number of multipliers, for each module as described in Table 2.9.

As described in Table 2.9, the forward kinematics modules (FK-HMD or FK-HSD) implemented using the fixed-point format [16.12] have saved 1.04×, 2.51×, and 2.75× of resources used for registers, LUTs, and multipliers, respectively when compared to the same implementation using floating-point representation. For the fixed-point [14.10] and [12.8] formats the number of multipliers also decreased 2.75×. In relation to the

Table 2.9: The difference of the amount resources of FPGA used by the floating-point implementation compared to the resources used by fixed-point implementations for different formats.

Module Name	Resources Name	Floating-Point F[32]	Reduction Rate of Area (*RA*) Fixed-Point [16.12]	[14.10]	[12.8]
FK-HMD	Number of Registers	3041	1.04×	1.05×	1.06×
	Number of LUTs	8008	2.51×	2.54×	2.61×
	Number of Multipliers	11	2.75×	2.75×	2.75×
KFF-HMD	Number of Registers	3113	1.06×	1.08×	1.09×
	Number of LUTs	12251	3.53×	3.89×	3.99×
	Number of Multipliers	48	3×	3×	3×
FK-HSD	Number of Registers	3041	1.04×	1.05×	1.06×
	Number of LUTs	8008	2.51×	2.54×	2.61×
	Number of Multipliers	11	2.75×	2.75×	2.75×
IK-HSD	Number of Registers	3149	1.07×	1.27×	1.66×
	Number of LUTs	14107	1.74×	1.87×	2.01×
	Number of Multipliers	27	3×	3×	3×
FBF-HSD	Number of Registers	323	2.24×	2.56×	2.99×
	Number of LUTs	1236	19.31×	20.94×	27.46×
	Number of Multipliers	9	3×	3×	3×

consumption of registers, the fixed-point [14.10] and [12.8] formats provided a reduction of 1.05× and 1.06×, respectively. The LUTs consumption presented for the fixed-point implementations showed a hardware usage 1.05× lower for the [14.10] format and 1.05× lower for the [12.8] format.

The fixed-point [16.12], [14.10] and [12.8] implementations of the kinesthetic feedback force (KFF-HMD) module showed a reduction for registers of 1.06×, 1.08× and 1.09×, respectively regarding the same implementation using floating-point representation. The LUTs resource reduction for these modules were of 2.51×, 2.54× and 2.61×.

For the inverse kinematics module (IK-HSD) implemented using fixed-point the use of registers, LUTs, and multipliers also decreased, as described in Table 2.9. In relation to the consumption of registers presented in Table 2.9, the formats [16.12], [14.10] and [12.8] have used 1.07×, 1.27×, and 1.66× less resources. The LUTs resource reduction for these modules was of 1.74×, 1.87× and 2.01×.

As described in Table 2.9, the feedback force module (FBF-HSD) implemented using the fixed-point format [16.12] have saved 2.24×, 19.31×, and 3× of the resources used for registers, LUTs, and multipliers, respectively when compared to the same implementation using floating-point representation. For the fixed-point [14.10] and [12.8] formats the number of multipliers also decreased 3×. In relation to the consumption of registers, the fixed-point [14.10] and [12.8] formats had a reduction of 2.56× and 2.99×, respectively.

The LUTs consumption presented for the fixed-point implementations showed a lower spend of 20.94× for the [14.10] format and 27.46× for the [12.8] format.

As shown in Table 2.9, the reduction achieved for the use of registers was lower, approximately 1× in almost all modules of hardware. Except for the FBF-HSD module which achieved a reduction of 2.99×. The area reduction related to LUTs was significant for the FBF-HSD module with a saving of hardware resources of approximately 27× for the 12-bit implementation. The FK-HSD and FK-HMD modules achieved reductions of approximately 2.6× for the implementations. The implementations of the KFF-HMD module achieved the best use of resources reaching approximately a 4× reduction for the 12-bit implementation. Implementations of the IK-HSD module showed little reduction in resources due to the usage of floating-point operations. Regarding the consumption of multipliers, all the implementations achieved very close reductions of approximately 3×.

Given these comparisons, the fixed-point representation of the modules has been able to optimize FPGA resource utilization while maintaining a level of accuracy within the acceptable range for the haptic device control application type. Thus, it can be affirmed that floating-point implementations require larger amounts of FPGA resources than an equivalent fixed-point solution. Due to higher resource usage, higher power consumption occurs.

2.8 Conclusion

This section presented two hardware proposals that make it possible to reduce the total latency produced by haptic devices that are inserted in a tactile internet environment. The main hardware strategy used in both proposals was to use reconfigurable computing (in FPGA) to minimize the execution time of the algorithms associated with the device. The first hardware proposal was designed using a 32-bit floating-point representation and the second hardware proposal was designed using a fixed-point representation with different formats (16-bit,14-bit and 12-bit). For both proposals, reconfigurable hardware reference models were designed for four modules that implement algorithms associated with robotics. The FK-HMD and FK-HSD modules implement the forward kinematics, the IK-HSD module implements the inverse kinematics, the KFF-HMD module implements the kinesthetic feedback force and the FBF-HSM module implements the feedback force. The modules were designed using a full-parallel implementation that works on a hybrid scheme that uses fixed-point and floating-point representation in distinct parts of the architecture.

Compared to the state of the art, this work stands out by presenting the description and implementation of four robotics algorithms in FPGA. The implementations of the modules (floating-point and fixed-point) presented in this work achieve higher module processing speed when compared to equivalent implementations from the state-of-the-art. All of the modules presented were analyzed based on the synthesis results, which included the hardware occupation area, sampling rate, and throughput.

Based on the synthesis results, it was observed that the implementations achieved high module processing speed, far below the latency limit of 1 ms. The designed hardware that uses 32-bit floating-point numeric representation achieved an acceleration of 93 ×

compared to the 37.5 μs time constraint. And the designed hardware that uses 16-bit fixed-point numeric representation achieved an acceleration of 165 $\times$ compared to the 37.5 μs time constraint. It is known that the floating-point is the most used data type to ensure that the calculations performed by the modules are highly accurate. However, in this work, the implementations presented similar behavior for both, floating-point and fixed-point architectures. Thus, it is more advantageous to use fixed-point architecture because was noted that reduced hardware occupancy and increased throughput.

It is possible to conclude that using reconfigurable embedded systems on devices such as FPGAs enables the parallel implementation of algorithms thus speeding up the processing of the data and minimizing execution time. The runtime gains obtained by the implementations presented in this work have allowed the use of critical applications that require short time constraints. As well as speeding up the processing, the low consumption of the occupancy area of the implementations developed here is a notable feature. This makes it possible for other systems to also be embedded in the FPGA, as is the case with prediction techniques, which can further assist in decreasing the latency of the tactile environment.

Chapter 3

General Purpose Processor for Tactile Control

This chapter aims to present, analyze and evaluate a reference model of a tactile glove device implemented in a microprocessor system that delivers tactile responses in a virtual environment. Details of hardware and software designs of a tactile glove, as well as the virtual environment, are described. Results and analysis about round trip latency time in the tactile internet environment are developed and compared with other works in the literature with the objective of validating the performance of the proposed tactile glove, which stands out as having the overall best performance against the ones it was compared to.

3.1 Introduction

Normally, on the tactile system, there are three main elements: the master device, the network, and the slave device. Depending on the type of device that is inserted into the environment, the mode of operation between elements may change. The teleoperated mode is also known as human-to-machine (H2M). In this system, the master device (local device) is controlled by a human operator and the slave device (remote device) is a robotic system (Aijaz 2016). In some cases on machine-to-machine (M2M) systems, the master device can be controlled by a robot where there is not a human in the loop (Yogeswaran et al. 2015, Dahiya et al. 2010)

The network is responsible for providing the infrastructure for transferring both haptic information data, kinesthetic, and tactile data between the devices. The communication from the master device to the slave device is called “direct communication” and, it is similar to the telecontrol system. The communication from the slave device to the master device is called “feedback communication” and, it is responsible for transmitting the tactile data that contains the information about sensations (weight, touch, vibration, temperature, and others) or kinesthetic data that contains the information about force (Aijaz et al. 2016, Simsek et al. 2016*b*, Fettweis 2014).

Tactile internet is an emergent topic and several researchers have been working on this subject. Inside this context, the works with tactile devices are fundamentals because they are main pieces on the tactile system (Dohler & Fettweis 2015, Oballe-Peinado et al. 2009, Szabo, Gulyas, Fitzek & Lucani 2015, Aristidou & Lasenby 2010, Culjat et al. 2010). As presented in (Aijaz et al. 2016, Simsek et al. 2016*b*, Fettweis 2014), the master and slave device must be designed on dedicated hardware with embedded systems, because they need to capture signals from sensors and they need to generate signals to actuators.

Another important point is about the algorithms associated to the tactile system, because the embedded system can run complex algorithms related to rotation matrices, matrix transformation, matrix product, non-linear functions, etc. The low processing hardware devices like microcontrollers cannot execute these algorithms in the time restrictions, about 0.15 ms for all processing. There are several kinds of tactile devices such as tactile gloves, robotic arms, exoskeleton hands, kinesthetic haptic device, and others (Dohler & Fettweis 2015, Oballe-Peinado et al. 2009, Ohnishi 2010).

The round trip communication between tactile devices and networks must have a latency within limits presented in the literature. In these conditions, this work contributes by presenting a novel embedded design and development of a tactile system that has low latency in communication between devices, respecting the time constraints in the millisecond interval. The tactile system designed has two main devices, a tactile glove, and a virtual environment. In addition, this work presents a comparison with other embedded systems applied to the tactile systems.

The tactile glove created enables the capture of kinematic actions from an operator's hand and also transmits tactile information to him from virtual objects through tactile feedback. In the virtual environment, in addition to a virtual hand, several virtual objects with different characteristics are created. The operator can control the virtual hand; in the virtual environment, the kinematic equations are implemented according to the characteristics of the hand model. A vital feature of this environment is that the operator wearing the tactile glove can remotely control the virtual hand and receive tactile information that represents the touch on the virtual objects. This tactile information may represent different types of materials and textures depending on the type of object that has been virtualized. Given the characteristics presented, the tactile system could be used on several applications such as telemedicine, remote diagnostics, games, remote analysis of materials, and others in which objects could be virtualized.

3.2 Related Work

In applications involving interactions in virtual environments as well as robot teleoperation applications (human-to-machine), it is necessary to artificially create a sense of touch or force for the operator to be stimulated. From these stimuli, characteristics of objects such as force, texture, weight, and temperature, for example, can be understood by the operator through the received sensation and thus a certain realism can be achieved. To provide this realism, tactile devices are used. As shown in (Culbertson et al. 2018), these devices are divided into three categories, being they graspable, touchable, and wearable. The graspable type devices are characterized by being kinesthetic systems, that is, they have force feedback. Touch-sensitive devices are systems that use displays that allow the operator to actively explore the entire surface. Wearable devices are typically characterized by being tactile (cutaneous) systems (Culbertson et al. 2018), but it is also possible to find proprioceptive systems (McCaw et al. 2018). Usually, these devices are mounted on the hands or other parts of the body that transmit sensations directly into the skin. As described in (McCaw et al. 2018), this wearable device is used to convey sensations and for the most part, they are developed in the form of gloves.

However, depending on their architecture, some kinds of gloves can provide both tactile and force feedback. Gloves that transmit force feedback are usually of the exoskeleton type as presented in (Fang et al. 2009) and (Ben-Tzvi & Ma 2014), these gloves are made up of mechanical parts that are required to provide force feedback. This type of glove is widely used in the rehabilitation and care of people who have some kind of disability (Popescu et al. 2013, Mohammadi et al. 2018). Due to the mechanical features aimed at providing the feeling of strength, with this device, it is generally not possible to feel object textures. Tactile gloves are used for this purpose.

Tactile gloves usually differ in the way they detect the movement of the operator's fingers, arms, and hand. In other words, they may have several degrees of freedom (DoF). Some works have a variety of ways to capture movements. In (Yang & Choi 2018) a camera is used to detect the movement of the fingers, already in (D'Abbraccio et al. 2019) a device called LeapMotion is used to capture the movement of the hand and arm. In the same context, the paper presented in (Sano et al. 2016) shows a rehabilitation system using a virtual reality system with sensory, visual, and auditory feedback. The operator interacts with virtual objects across multiple devices. Arm detection is captured by a Kinect-type human motion detection system and hand movement and finger flexion is captured through a CyberGlove® II type glove. The defined environment allows very realistic local interaction between the operator and the environment due to the devices used. However, its architecture uses proprietary equipment such as the Kinect, the CyberGlove. The use of these devices may limit the replication of this experiment as they depend on specific hardware.

Another way to capture operator finger and arm positioning is through the use of inertial measurement unit (IMU) sensors as shown in (Lin et al. 2018, Lobo & Trindade 2013, Arjun et al. 2018, Weber et al. 2016). These sensors allow capturing some kinematics of the hand, including the fingers and forearm. If compared to previous works presented, the use of IMUs sensors can make developing a glove cheaper and easier to replicate. However, the application needs and the development can be complex according to the amount of DOFs to be captured.

When there is no possibility of using sensors to capture the hand movements, the uses of predefined stimuli can assist in the development of applications with tactile actuators. With the glove device presented in (Muramatsu et al. 2012), it is possible to receive tactile sensations of virtually emulated objects. The glove receives stimuli locally from a tactile information generator server. The stimuli are predefined and sent to the operator without interaction between them. This approach can be useful for validating the types of textures and materials that will be used.

On the other hand, the use of gloves that have only sensitivity sensors can help the way that materials and textures of real objects can be represented virtually. In the works, (Sagisaka et al. 2011) and (Sagisaka et al. 2012), two types of high-density tactile detection gloves with 1052 sensitivities elements are proposed. The proposed gloves allow pressure measurement at 1052 points in a human hand. Due to this amount of points, it is possible to detect very small real objects in almost every part of the hand. For the models presented in (Sagisaka et al. 2011) and (Sagisaka et al. 2012), the gloves are limited only in capturing the information about the touch of the hand sensors with some type of

object, thus differing from the model presented in this work which presents a glove with actuators.

In the context of the tactile internet, artificial skinned humanoids can replace the human operator or even be used as an artificial member of a human operator, enabling exchange information with another type of robot performing the M2M communication. As presented in (Dahiya 2019), estimates of contact parameters such as force, soft contact, hardness, texture, and temperature, among other features can be detected by a robot. However, the development of artificial skin can be complex depending on the level of similarity to human skin.

The authors in (Ulmen & Cutkosky 2010) proposed a low-cost artificial robot skin that could be used to capture tactile touch. With the received data from artificial skin it is necessary to find out the type and the characteristics of the touched material. Some works in the literature discuss how robots can recognize the types of materials and their characteristics. In (Yao et al. 2017), the authors show how the center of mass of real objects can be obtained. The works (Kaboli & Cheng 2018) and (Kaboli et al. 2015) have presented solutions for the recognition of objects through surface textures, it was presented in the methodology that the recognition rate of textures and objects was above 90%. Based on work (Shirafuji & Hosoda 2014), it is possible to understand how to control the force exerted by a robot's hands based on the grasp force, as well as to detect the slip of objects.

When a robot starts to perceive the characteristics and properties of an object it may be able to identify it. However, depending on the varying characteristics of the known object (material, texture), there is a possibility that it will not be identified. To enable the identification of variations of object characteristics known by the robot, the authors in (Feng et al. 2018) and (Kaboli et al. 2018) presented algorithm models that aimed solving this problem. The work (Kaboli et al. 2019) presents a robot capable of identifying unknown objects by their physical properties (surface texture, stiffness, and thermal conductivity).

Among the works presented, those that focus on machine-to-machine applications ((Sagisaka et al. 2011, Sagisaka et al. 2012, Ulmen & Cutkosky 2010, Yao et al. 2017, Kaboli & Cheng 2018, Kaboli et al. 2015, Shirafuji & Hosoda 2014)), are more focused on the development of devices with sensors for texture detection and recognition. This is different from the proposal of this work, which is focused on the development of a tactile glove with IMU sensors and vibration actuators that are activated when there is some kind of interaction with virtual objects.

In the human-to-machine system line, the architectures presented in the works (Lobo & Trindade 2013, Arjun et al. 2018, Weber et al. 2016) allow the tactile glove to handle real robotic systems. However, when there is no physical model, a new architecture must be developed. Another important point is that in these environments, textures and virtual objects cannot be felt.

As can be seen from the works (Lobo & Trindade 2013, Arjun et al. 2018, Weber et al. 2016), gloves differ in design and some features. For example, the manner in which the position of the fingers, hand, and arm is captured. Another point is how the glove communicates with the controlled device. It is also important to emphasize that in none of these works is it possible to perform a glove interaction with virtual objects, only the work

(Muramatsu et al. 2012) allows the reception of already predefined stimuli. Therefore, in this work, a complete environment is proposed so that operators with the tactile glove can interact and feel textures remotely from a virtual environment. Unlike the work presented in this proposal, a complete specification of the environment will be provided, both the glove design and the electronics, as well as the virtual model.

3.3 System Architecture

The high level block diagram presented in Figure 3.1 presents an overview of the envisaged scheme which represents the tactile system. The scheme basically has a local device (known as master) and a remote device (known as slave) that communicate over the internet through a bidirectional data communication network. The master device is a tactile glove which is controlled by an operator and the slave device is a personal computer showing a virtual robotic manipulator.

As can be seen in Figure 3.1, the operator wearing the tactile glove can remotely control a robotic manipulator to do the desired task. The initial step is identified as the movement which the operator performs when wearing the glove. These movements are detected by the sensors present on the glove and sent to the computer to control the virtual robotic manipulator. The second step is identified as the data communication network between the master and slave devices. This network is connected to the Internet is usually composed of transmitters, routers, switches, and other communication components. The subsequent step is identified as the steps performed by the personal computer so that the virtual robotic manipulator performs the movements sent by operator. In this stage, the collision and feedback control are generated so that stimuli are sent to the operator. The final step is identified as the result of the process of operator interaction with the virtual environment. In this step, the feedback signals can be received by the actuators present in the glove to transmit the vibrotactile sensation to the operator.

To better understand the steps presented, Figure 3.2 shows the general proposed architecture scheme which represents the tactile system. The proposed model is formed by four subsystems called operator (OP), tactile glove (TG), network (NW) and virtual environment (VE). The tactile glove is equipped with sensors and actuators that allow the operator to interact and manipulate objects that are inserted into a virtual environment, aiming to perform some type of task. Data communication between the tactile glove and the virtual environment occurs through the network.

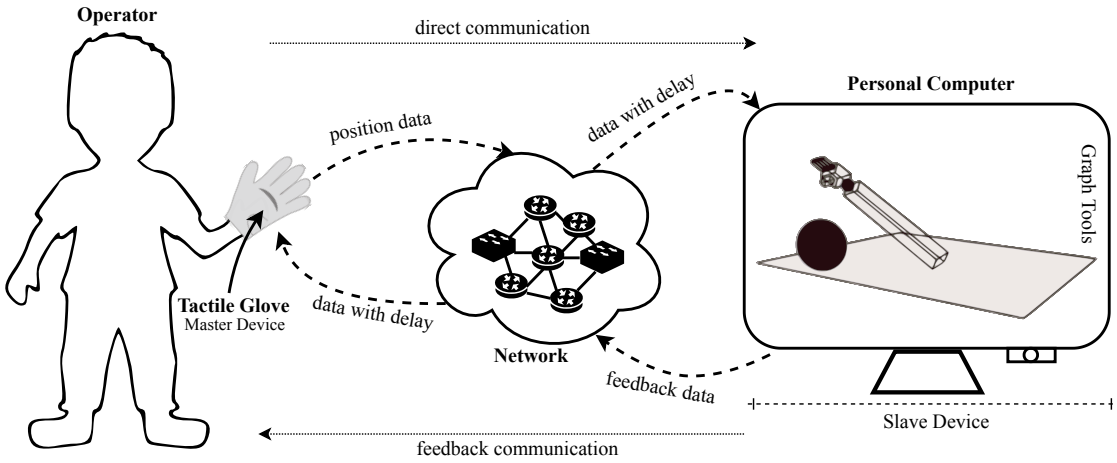


Figure 3.1: High-level block diagram of the human-to-machine tactile system.

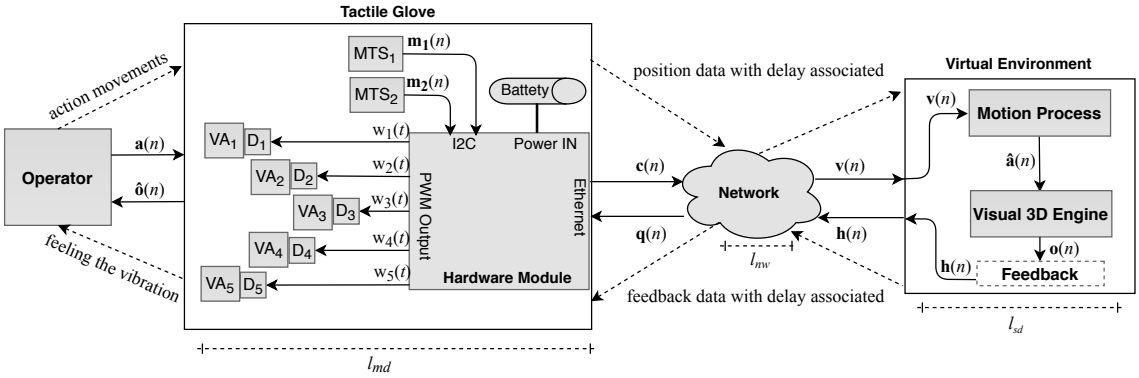


Figure 3.2: Block diagram of the human-to-machine tactile system architecture.

The tactile glove is composed of two motion tracking sensors, called here MTS_1 localized on the hand and MTS_2 localized on the arm, five vibration actuators localized on the fingers, called VA_i, where $i = 1 \ldots 5$ and a hardware module, five drivers, called D_i, where $i = 1 \ldots 5$ and battery. The network provides an infrastructure to transmit signals from the operator to the virtual environment and feedback signals in the reverse direction. The virtual environment is composed by a PC running a virtual engine 3D.

When an operator is using the tactile glove he can begin to interact with the virtual environment. As shown in Figure 3.2, the signal $\mathbf{a}(n)$ represents the kinematics movement performed by the operator. When the operator carries out some type of kinematic movement, the j-th MTS modules present on glove hardware compute the resulting position of operator movement in terms of quaternions at each n-th instant, and send this information through the discrete signal by a vector $\mathbf{m}_j(n)$ expressed as

$$\mathbf{m}_j(n) = \begin{bmatrix} qw_j(n) \\ qx_j(n) \\ qy_j(n) \\ qz_j(n) \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\theta(n)}{2}\right) \\ vx_j(n)\sin\left(\frac{\theta(n)}{2}\right) \\ vy_j(n)\sin\left(\frac{\theta(n)}{2}\right) \\ vz_j(n)\sin\left(\frac{\theta(n)}{2}\right) \end{bmatrix} \tag{3.1}$$

where $[\theta, vx, vy, vz]$ are the four parameters that define the quaternion. $\theta(n)$ is the angle of rotation and vx, vy, and vz represent the axis of rotation.

As shown in Figure 3.2, after the hardware module receives the $\mathbf{m}_1(n)$ and $\mathbf{m}_2(n)$ signals through the I2C communication protocol it creates a new discrete signal by a vector to be sent to the network. The newly created signal $\mathbf{c}(n)$ is expressed as

$$\mathbf{c}(n) = [\mathbf{m}_1(n), \mathbf{m}_2(n)] \tag{3.2}$$

where $\mathbf{m}_1(n)$ is the quaternion information about module MTS_1 and $\mathbf{m}_2(n)$ is the quaternion information about module MTS_2.

When the signal $\mathbf{c}(n)$ sent by the tactile glove is transmitted and propagated through the network to the virtual environment, this signal can have some type of disturbance. So when the network receives the signal, a delay here called d^f is considered, resulting in a new signal $\mathbf{v}(n)$ which is expressed as

$$\mathbf{v}(n) = [\mathbf{m}_1(n - d^f), \mathbf{m}_2(n - d^f)] \tag{3.3}$$

where $\mathbf{m}_1(n - d^f)$ and $\mathbf{m}_2(n - d^f)$ are the data transmitted by the network with a delay d^f at the n-th instant of time.

As soon as the $\mathbf{v}(n)$ signal arrives in the virtual environment, it is directed to the motion process module which is responsible for processing information related to the movements in the virtual environment.

Then, with the quaternion information received through the $\mathbf{v}(n)$ signal, it is possible to determine the angular vector of rotation, also called the Euler angles associated with the tactile glove. Thus, the signals containing the quaternions $\mathbf{m}_1(n)$ and $\mathbf{m}_2(n)$ are transformed into Euler angles so that the positioning of the hand (MTS_1) when the arm (MTS_2) is determined. This process is performed every n-th instant and sent to the visual 3D engine module via signal discrete $\hat{\mathbf{a}}(n)$ which is expressed as

$$\hat{\mathbf{a}}(n) = \begin{bmatrix} \phi_j(n) \\ \theta_j(n) \\ \psi_j(n) \end{bmatrix} = \begin{bmatrix} \arctan\left(\frac{2(qw_j(n)qx_j(n)+qy_j(n)qz_j(n))}{qw_j(n)^2-qx_j(n)^2-qy_j(n)^2+qz_j(n)^2}\right) \\ -\arcsin\left(2(qx_j(n)qz_j(n)-qw_j(n)qy_j(n))\right) \\ \arctan\left(\frac{2(qw_j(n)qz_j(n)+qx_j(n)qy_j(n))}{qw_j(n)^2+qx_j(n)^2-qy_j(n)^2-qz_j(n)^2}\right) \end{bmatrix} \tag{3.4}$$

where $j = 1 \ldots 2$ represents the MTS_j values, $\phi_j(n)$, $\theta_j(n)$ and $\psi_j(n)$ are called the yaw, pitch, and roll, respectively.

At the moment the $\hat{\mathbf{a}}(n)$ signals are received by the visual 3D engine, it is possible to calculate the current glove position in space, expressed by vector $\mathbf{s}_j(n) = [s_j^x(n), s_j^y(n), s_j^z(n)]$ for hand and arm through kinematic calculations or through calculations using rotational matrices. Thus, after performing these calculations it is possible to display the positioning of the tactile glove in a virtual way. To do this, the application created in visual 3D engine that implements the virtual model of the manipulator performs the positioning of the hand and the arm every n-th instant.

After the virtual manipulator begins to move, it can find some virtual objects in the way. Virtual objects are also created in the visual 3D engine; they can be made with different types of materials and textures. When the operator virtually touches objects, the collision detection routines are triggered to generate some kind of stimulus. The touch sensation is sent from the virtual environment to the operator via tactile feedback.

When the virtual tactile glove moves in the environment at every n-th instant, the equation responsible for detecting the collision is performed, it is expressed as

$$r(n) = \frac{(N_a s_j^x(n) + N_b s_j^y(n) + N_c s_j^z(n)) - (N_a x0 + N_b y0 + N_c z0)}{\sqrt{N_a^2 + N_b^2 + N_c^2}} \tag{3.5}$$

where $\mathbf{N} = [N_a, N_b, N_c]$ is a normal vector and $x0$, $y0$, and $z0$ is the position of the virtual object.

After the collision routines are executed, if a touch is detected then the routine responsible for generating the tactile feedback is triggered. The tactile feedback routine is based on the spring-damper force model as presented in (Yang et al. 2018*b*). The tactile feedback for each i-th VA_i is obtained by the equation expressed as

$$f_i(n) = k_i r(n) \tag{3.6}$$

where $i = 1 \ldots 5$ and k_i is the i-th spring constant.

Then the feedback information (about sensation) is sent to the master device through the discrete signal of a vector that can be expressed as

$$\mathbf{h}(n) = [f_1(n), \ldots, f_5(n)] \tag{3.7}$$

where $f_i(n)$ is the signal associated of the i-th finger at the n-th instant of time. The signal

$f_i(n)$ can be a value between zero and 100. These values can be changed according to the type of force exerted on the virtual object.

As shown in Figure 3.2, the $\mathbf{h}(n)$ signal that is associated with feedback information is sent from virtual environment to tactile glove through the network. As previously stated, the signals transmitted by the network may suffer perturbations, so a new discrete signal by a vector $\mathbf{q}(n)$ is created being expressed as

$$\mathbf{q}(n) = [f_1(n-d^b), \ldots, f_5(n-d^b)] \tag{3.8}$$

where $f_i(n-d^b)$ is the data transmitted by the network with a delay d^b at the n-th instant of time.

Thereafter, the hardware module on the tactile glove receives the $\mathbf{q}(n)$ signal and calls the routines responsible for providing feedback to the operator. The technique consists in varying the working time for each i-th VA_i actuator, where it increases according to the pressure exerted on the virtual object. Each vibration actuator, VA_i, was governed by a driver, D_i, using a pulse width modulation (PWM) signal, $w_i(t)$, expressed as

$$w_i(t) = \begin{cases} a & \text{if } \frac{f_i(n)}{100} > c(t) \\ 0 & \text{if } \frac{f_i(n)}{100} \leq c(t) \end{cases} \tag{3.9}$$

where a is the amplitude of the signal, $f_i(n)$ is the pulse width, which varied from 0 to 100%, and $c(t)$ is a sawtooth signal with amplitude 1 and frequency f_{PWM}. The driver, D_i, regulated the voltage at the terminals of the VAs according to

$$v_i(t) = f_i(n) v_a^{max}(t) \tag{3.10}$$

where $v_a^{max}(t)$ is the maximum voltage at the terminals of the each i-th actuator VA_i.

At the end of the process, with PWM techniques it is possible to change the vibrations so that the glove produces tactile stimulation through the actuators, as can be seen in Figure 3.3. The wavelength of the virtual surface can be modified at each instant of time so that the operator feels vibrations that inform them about the object they are manipulating.

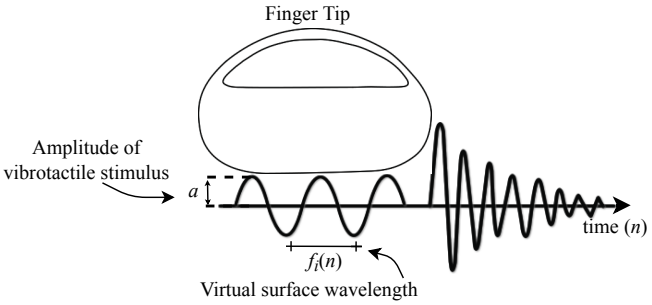


Figure 3.3: The vibrotactile stimulus for sensations.

3.4 Description of the Design

This section includes hardware and software design of the system architecture shown in Figure 3.2.

3.4.1 The Tactile Glove

Embedded System

The embedded system was developed for Intel Galileo 2nd Generation. This microcontroller board is based on the Intel® Quark SoC X1000 application processor of the 32-bit Pentium class.

In the glove device each j-th MTS is the MPU-6050, where each IMU contains a digital motion processor (DMP) which fuses the accelerometer and gyroscope data together (Six-Axis Gyro + Accelerometer). The MTS-1 is localized on the wrist and the MTS-2 is localized on the center of the hand. With the two MTS it is possible to obtain the spatial localization on the hand and the forearm at the n-th instant of the time.

In addition, there are the five vibration actuators; VAs provides tactile feedback to the glove. To provide the sensation of vibration, the eccentric rotating mass motor (ERM) is used. Each j-th is an ERM and vibrates when a DC voltage is applied across it. The VA_1 is located on the lower tip of the thumb, the VA_2 is located on the index, the VA_3 is located on the middle, the VA_4 is located on the ring, and the VA_5 is located on the pinky (see Figure 3.4).

As shown in the Figure 3.2, between hardware module and each i-th VA_i there is a driver circuit D_i. The driver circuit associated to each VA_i is shown in Figure 3.5.

This circuit is composed by two resistors, R_1 and R_2, with values 390 Ω and 4.7 Ω respectively, one optoacoplador 4N35, one transistor NPN 2N3904, one rectifier diode 1N4007, and a battery.

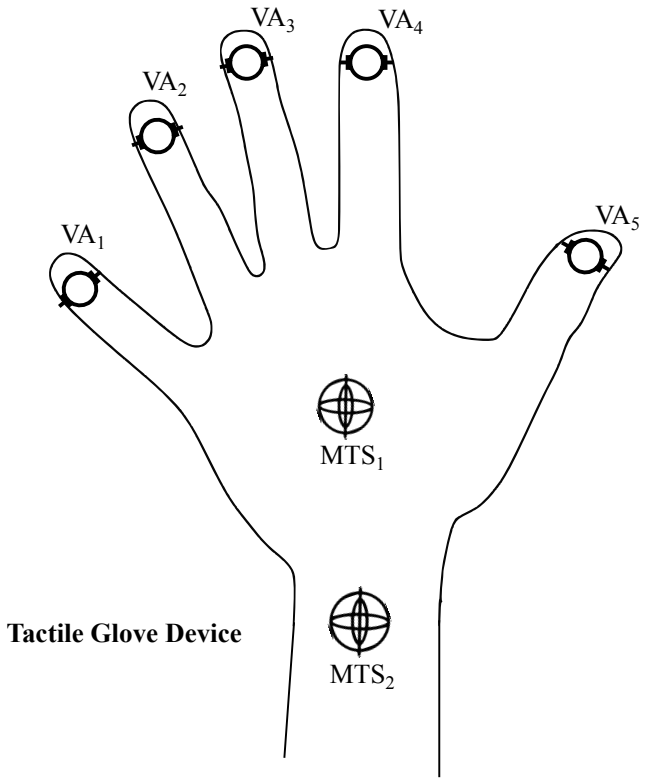


Figure 3.4: Position of sensors' motion tracking sensors (MTSs) and actuators' vibration actuators (VAs) in tactile glove.

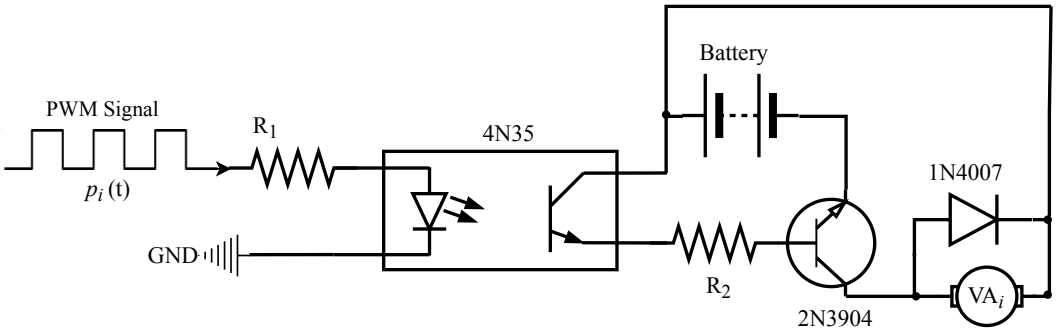


Figure 3.5: The driver circuit associated to each vibration actuator, VA_i.

Software

The Hardware module is running Linux Yocto and there is an embedded application in C++ called here MasterApp. Two important libraries are used to carry out this process. The first one is used to provide I2C communication between MasterApp and each j-th MTS (shown in Figure 3.2), for that was used i2cdevlib. The second library is used to control GPIOs and for that is used a low-level C/C++ library called MRAA. This library

provides bindings in a few programming languages to I/O interface in several hardware platforms. Using this library it is possible to create a code that is compatible with various hardware platforms. That is, its use is not tied to a specific hardware.

The steps processed by the MasterApp are presented in Algorithm 1 and are described in detail below.

In the first step (line 1 of Algorithm 1), the MasterApp initializes the variables and libs. After that, the MasterApp stays in a loop until the simulation is stopped. When the simulation is in progress, it is verified if the I2C connections to the MTS are working properly. If everything is working, the steps presented in lines 4, 5, and 6 are performed.

In the steps shown in lines 4 and 5, the MasterApp captures quaternion information from each j-th MTS through I2C communication. Being that the variable $\mathbf{m}_1(n)$ receives the information from address 0X68 and $\mathbf{m}_2(n)$ from address 0X69.

As described in Equation (3.1) each signal $\mathbf{m}_j(n)$ is composed of four pieces of information. Thus, a new variable $\mathbf{c}(n)$ (as presented in line 6 and according to Equation (3.2)) containing a packet of information composed by $\mathbf{m}_1(n)$ and $\mathbf{m}_2(n)$ is created to be sent to SlaveApp through a TCP socket. If everything is doing right, the variable $\mathbf{c}(n)$ is sent to SlaveApp as described in line 9 of Algorithm 1.

After these steps the information about feedback $\mathbf{h}(n)$ can be received, according to the step of line 11. With the values obtained by signal $\mathbf{h}(n)$, it is possible to generate the sensation of tactile feedback through the PWM modulation. For that the step as shown in line 12 must be processed according to Equation (3.9). After that, the MasterApp performs the next loop.

Algoritmo 1: MasterApp (Glove Device Algorithm)

```
1  initialization;
2  while simulationIsRunning do
3      if isDMPReady then
4          m1(n) ← getQuaternionFromI2C(0x68);
5          m2(n) ← getQuaternionFromI2C(0x69);
6          c(n) ← generateQuaternionPacket(m1(n), m2(n));
7      end
8      if hasIMUData then
9          sendQuaternionToSlave(c(n));
10     end
11     h(n) ← readFeedbackDataFromNetwork();
12     w(t) ← generateFeedbackSensationWithPWM(h(n));
13 end
```

3.4.2 The Network

The network is the communication medium used to transmit the tactile glove's actuation signals to the virtual environment as well as the feedback signals sent from the virtual environment to the glove. Applications that are in the context of the tactile Internet often require a network environment that has very low latency. However, since the main

purpose of this work is the development of the tactile glove architecture and not of the network environment, it is assumed that the use of a local network routing device satisfies this requirement. Therefore, an Askey RTF3505VW-N2 router model was used to enable the tactile glove and the virtual environment to communicate over a wired LAN network with connection on the internet.

3.4.3 The Virtual Environment

The virtual robotic hand was modeled using the software Processing 3. Processing is a flexible software with an easy language for development of virtual environments. The steps processed by the SlaveApp are presented in Algorithm 2 and are described in detail below.

Algoritmo 2: SlaveApp (Virtual Environment Algorithm)

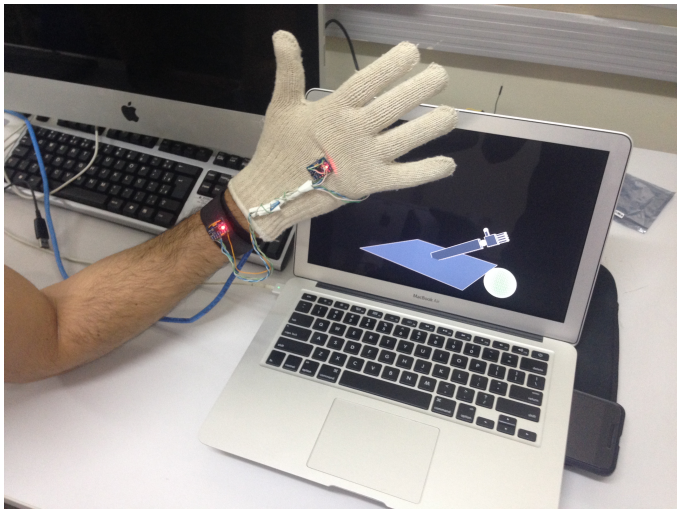
```
1 initialization;
2 while simulationIsRunning do
3     v(n) ← readQuaternionFromNetwork();
4     â(n) ← getEulerAngles(v(n));
5     moveVirtualArmHand(â(n));
6     h(n) ← detectCollision();
7     sendFeedbackToMaster(h(n));
8 end
```

In the first step (line 1 of Algorithm 2), the SlaveApp initializes the variables and libs. After that, the SlaveApp stays in a loop until the simulation is stopped. When the simulation is in progress, the SlaveApp receives $\mathbf{v}(n)$ as quaternion packet information from MasterApp through TCP socket, as shown in step in line 3. In the next step the variable $\hat{\mathbf{a}}(n)$ is obtained as shown in line 4 after transformate quaternion packet in euler angles. Posteriorly moves the virtual robotic hand.

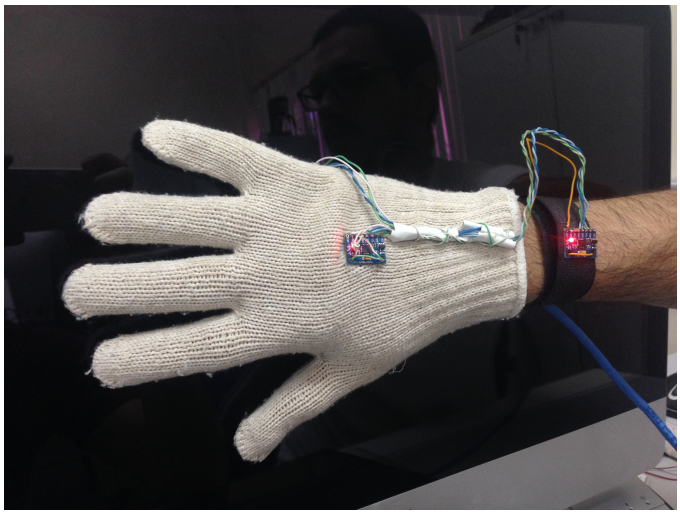
While the hand is moving around the environment, any collision on objects created in the virtual environment can be detected the step shown in line 6. Finally the data $\mathbf{h}(n)$ about feedback is sent to MasterApp. After that, the SlaveApp perform the next loop.

3.5 Results

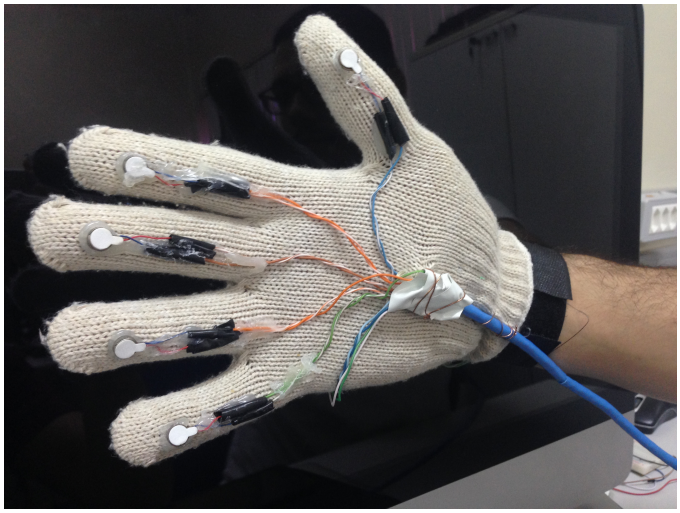
The final result of the proposed glove can be seen in illustrations presented in Figure 3.6. In the illustration of Figure 3.6a, it is possible to observe the tactile glove (master device) controlling the virtual environment (slave device). In the illustration Figure 3.6b are the sensors MTS_1 and MTS_2. Finally, in the illustration of Figure 3.6c, it is possible to observe the five vibrotatile actuators' VAs.



(a)



(b)



(c)

Figure 3.6: The final result of the design of the proposed glove. (**a**) Tactile glove and slave device (PC with virtual robotic arm). (**b**) Position of inertial measurement units (IMUs) (motion tracking device) in the tactile glove. (**c**) Position of all fingers' actuators (five vibration actuators) in the tactile glove.

In Figure 3.7 the developed hardware used for controlling the tactile glove is presented. It contains the Galileo Gen2 board, the drivers, and the battery.

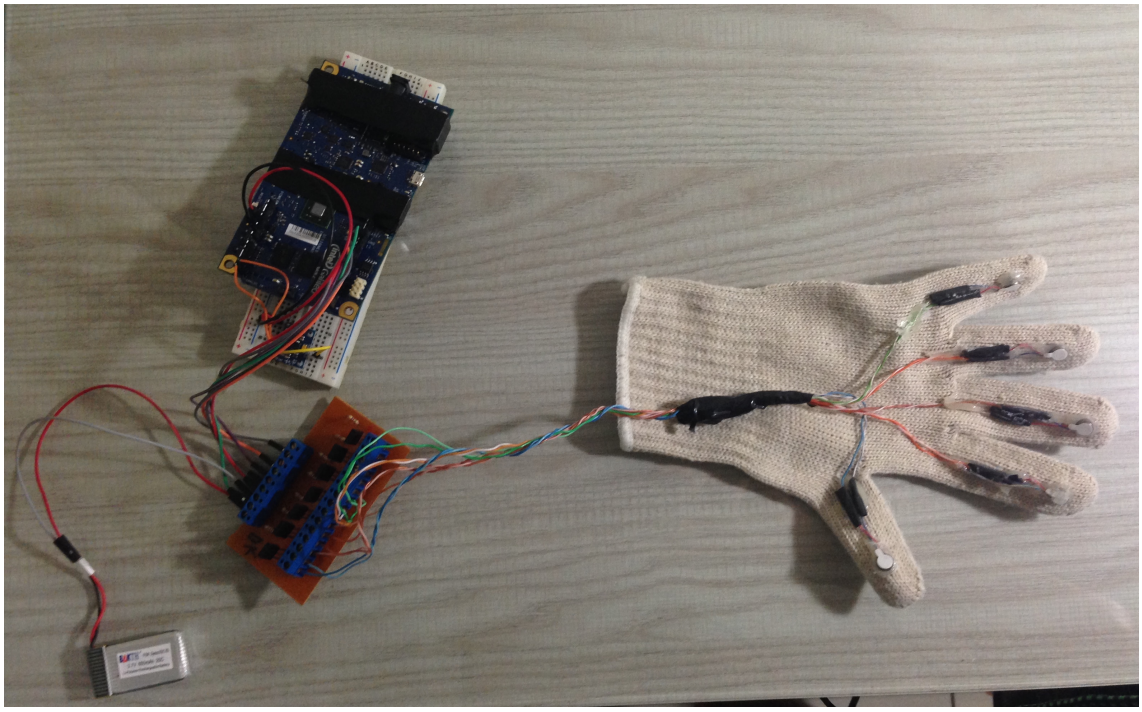


Figure 3.7: Final version of the hardware.

3.5.1 Round Trip Delay and Component Latencies

Based on this, a brief analysis of the delay of the modules involved in this work is carried out. Figure 3.8 provides an overview of the developed environment. It is possible to observe five steps that are performed so that the entire cycle of interaction between the tactile glove and the virtual environment is realized.

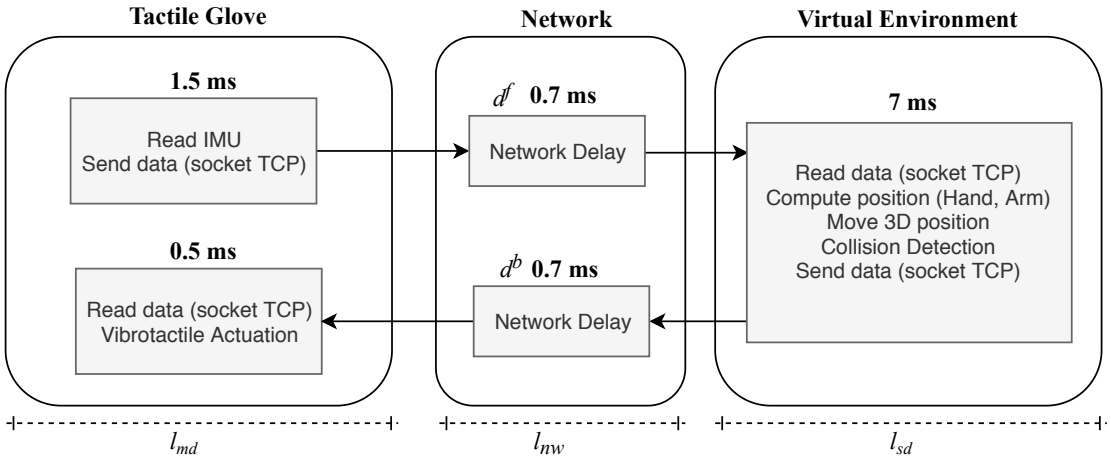


Figure 3.8: Component latencies. Round trip latency is 10.4 ms.

The first step is related to the delay spent by the glove device; it involves the process of reading the IMUs (MTS_1 and MTS_2) and sending the information through the TCP socket. These processes take 1.5 ms to be finalized.

The delay related to data transmission over the network is defined as d^f for when the signals are transmitted from the master device to the slave and d^b when the signals are transmitted in the reverse path. In the architecture used, the values for d^f and d^b are 0.7 ms and 0.7 ms respectively. Thus, the total latency spent by the network l_{nw} is given by the delays d^f and d^b, which has a total of 1.4 ms.

The step which composes the virtual environment involves the process of calculating the position of the hand—rendering of the positioning of the hand in the 3d environment—and the collision process, which also involves feedback. The total latency for this step is given by l_{sd}, which has a value of 7 ms.

The feedback sent by the virtual environment goes through the network again with the delay already shown. Finally, the hardware present in the tactile glove receives the feedback signals through the socket and generates the PWM signals for the vibrator actuators; this process generates a delay of 0.5 ms.

The total latency of the system is given by the sum of processing time spent on the glove added to the total transmission time plus the total processing time spent by the virtual environment. Thus, the total latency of the system was calculated by the equation expressed as

$$l_{total} = l_{md} + l_{nw} + l_{sd} \tag{3.11}$$

The latency obtained from the tactile glove l_{md} and the latency obtained from the virtual environment l_{md} is high due to the hardware model used. Another limiting point is the transmission rates between the components due to the communication protocols that were used.

The round trip latency of the environment was about 10.4 ms. Thus, with this obtained value it is possible to conclude that this application is within the requirements necessary to be used in tactile internet applications (Li et al. 2018, Nasrallah et al. 2018, Antonakoglou et al. 2018*b*).

Table 3.1: Comparison of the hardware, sensors, and actuators used in this work with other works.

Reference	Glove Hardware	MIPS/MHz/Core	Processor Bits	Sensors	Actuators
(Lin et al. 2018)	MSP430F5438A	-	16	18 IMUs	No
(Lobo & Trindade 2013)	FPGA DE0-nano	-	-	11 IMUs	14 Vibrotactile
(Arjun et al. 2018)	Cypress PSoC 5LP	1.25	32	5 IMUs	2 Vibrotactile
(Weber et al. 2016)	ATmega32U4	-	8	10 Flex + 1 IMU	5 Vibrotactile
(Muramatsu et al. 2012)	PIC	-	8	5 Flex	5 Vibrotactile
This work	Galileo Gen2	1.25	32	2 IMUs	5 Vibrotactile

Table 3.2: Comparison of our proposed glove and other gloves.

Reference	Mov. Detection	Feedback	Communication	Internet	Virtual Env.
(Lin et al. 2018)	Finger + Hand + Forearm	No	Bluetooth	No	No
(Lobo & Trindade 2013)	Finger	Yes	UART+Wifi	No	No
(Arjun et al. 2018)	Finger + Hand + Forearm + Arm	Yes	UART + PC	Yes	No
(Weber et al. 2016)	Finger + Hand	Yes	Bluetooth	No	No
(Muramatsu et al. 2012)	Finger	Yes	Bluetooth	No	Yes
This work	Hand+Forearm	Yes	TCP	Yes	Yes

3.5.2 Comparison with the Related Works

Table 3.1 shows a comparison of the related works. The first column presents the related works. The second, the hardware model that is integrated into each project. The third and fourth columns are related to the processor type used in the glove hardware, where the third shows information about MIPS/MHz/Core processing efficiency and the fourth, the number of processor bits. The last two columns show the number of sensors and actuators used on the sleeve respectively.

As can be seen from Table 3.1, there were variations in the type of hardware used in developing the gloves. The work presented in (Lobo & Trindade 2013) used an FPGA board, the works (Lin et al. 2018, Weber et al. 2016, Muramatsu et al. 2012) used microcontrollers with 16, 8, and 8 bits respectively. Only the work (Arjun et al. 2018) used a 32-bit microprocessor equivalent to what is being used in the proposal presented in this work. It can be noticed that all the related works presented in Table 3.1 used some type of sensors. The IMU was the one chosen in most of the projects and it is used to capture finger, hand, forearm, and arm movements. Unlike other works, the authors in (Weber et al. 2016, Muramatsu et al. 2012) used flexible resistive sensors to capture finger position. Regarding the actuators, only the work (Lin et al. 2018) did not use any. All others used vibrating actuators, differing only in the amount used.

Table 3.2 shows the comparison of other characteristics in relation to the same works that were presented in Table 3.1. In Table 3.2, The first column identifies each related work. Subsequently, the second column shows in which locations the sensors and actuators shown in Table 3.1 were allocated. The third column represents information about the use of tactile feedback. The fourth column is related to the type of communication that was used to communicate between the glove and the device. Finally, the last two columns present whether the proposed environment enables communication through the internet and if the developed architecture allows the glove to communicate with any virtual environment.

Table 3.3: Round trip latency and speedup measurement results.

Reference	Round Trip Latency	Speedup
This work	10.4 ms	-
(Arjun et al. 2018)	85 ms	8.17
(Weber et al. 2016)	40 ms	3.85

As can be observed in Table 3.2 only the glove proposed in this work has a TCP communication interface with the internet without the need for extra devices. The works (Lin et al. 2018, Weber et al. 2016, Muramatsu et al. 2012) only allow local communication via Bluetooth with the slave device. In work (Lobo & Trindade 2013), even though the glove has wifi connectivity, the environment does not provide an internet connection. The work (Arjun et al. 2018) allows an internet connection, however, the glove is dependent on a UART connection with a personal computer.

An important point in the proposal of this work is the interaction of the glove with virtual objects which allows the identification of different textures. As shown in Table 3.2, only work (Muramatsu et al. 2012) has a virtual environment, but the proposed environment does not allow communication over the internet and interaction is limited only to the reception of pre-defined sensations.

Table 3.3 presents the round trip and speedup measurement results of the related works. Among the works presented in the previous tables, only works (Lin et al. 2018, Arjun et al. 2018, Weber et al. 2016) present latency results of the developed environment. These works are listed in the first column of Table 3.3, in the second column, it is possible to observe the round trip latency. Finally, the last column presents the speedup obtained in relation to the proposal presented in this work with the references.

The work (Arjun et al. 2018) shows a round trip latency of 85 ms. Although the authors use a 32-bit microprocessor, high latency may be caused by the type of protocol used for communication between components. In work (Weber et al. 2016) the results indicate that the main loop of the application is executed with a frequency of 25 Hz. This value is equivalent to a 40 ms round trip latency.

The round trip latency values of the works (Arjun et al. 2018) and (Weber et al. 2016) are higher than the result obtained by the work here presented. As can be seen from Table 3.3, this work has a speedup of 8.17 times faster than (Arjun et al. 2018) and 3.85 times faster than (Weber et al. 2016).

At the moment, the main limitation of the prototype presented in this work is related to the number of existing actuators. However, this amount can be expanded. Additionally, the presented prototype can be improved by using dedicated hardware to speed up data processing. As a result, information execution time may decrease, and the round trip latency can be shorter.

3.6 Conclusions

This work presents a proposal for the implementation of a tactile glove and a virtual environment inserted in a tactile internet environment. The model gives the operator direct contact with the virtual objects, giving the impression that there is physical touch. With this, the operator can perceive what type of product or material is being touched, in addition to being able to feel different types of textures. The proposed model differs from the works presented in the literature since the interaction between the real and the virtual world is independent of the location of the master and slave devices. The glove and the virtual environment may be in the same place or even geographically distant because in the tactile Internet environment this limitation is solved. Full details of the architecture and implementation of the tactile glove as well as the virtual environment are provided with the aim of contributing to the development of new applications for the tactile Internet. The practical results obtained confirm that the model works as expected and demonstrate the viability of the application in tactile Internet environments.

Chapter 4

Conclusions

This work investigated and proposed the use of hardware strategies in order to minimize the execution time of the algorithms associated with haptic and tactile devices. As a first step, this thesis carried out a bibliographic study on the use of three hardware strategies in order to investigate the processing time spent by hardware modules in tactile internet applications. The first and second strategy was based on the use of reconfigurable computing, and the third was based on the use of microprocessed systems. It was found that most scientific papers available do not have a complete tactile internet environment, that is, wich involves kinematics and feedback modules remotely through bilateral communication and that respect the limits of latency restriction. In view of this, the innovative aspect of this thesis was to propose hardware reference models that have high processing. As well as presenting two types of applications inserted in two types of tactile internet environment that using the proposed hardware models are within the latency restriction limits.

In the first model of the architecture of the tactile internet environment that was presented, two robotic devices of the manipulator type with haptic functions were used as master and slave devices. For each device, there is associated hardware that is responsible for implementing the robotics algorithms: direct and inverse kinematics, feedback force, and kinesthetic feedback force. For this application, two hardware strategies that use reconfigurable computing (in FPGA) for the modules that implement the algorithms associated with the devices were presented. In the first hardware strategy, the circuits associated with the device modules were designed in FPGA using a floating-point numerical representation. In the second strategy, the modules were designed using fixed-point representation with different formats. In both proposals, the circuits process the algorithms in a completely parallel manner, causing several calculations to be performed at the same time.

In view of the implementation details of the circuits, as well as the results associated with the sampling rate, transfer rate, latency and post-synthesis occupation area that were presented, it was possible to see that the strategies used are capable of minimizing the execution time of the algorithms, as they achieve high throughput rates. However, the strategy of using fixed-point presented better results compared to the floating-point implementation. It was observed that the implementation in fixed-point had less occupation of the hardware resources and high throughput rates. In general, when compared to other implementations found in the literature, it is possible to confirm the superiority in performance due to the high acceleration rates.

In the second application model that was presented, the tactile internet environment allows an operator with a tactile glove to interact with objects inserted in a virtual environment. The tactile glove has the functionality to provide the operator with tactile feedback,

enabling direct contact with virtual objects, giving the impression that there is physical touch. The virtual environment artificially creates a tactile sensation so that the operator is stimulated when there is a touch on an object. For this application, the hardware strategy used was to physically design the tactile glove type device, where the sensors and actuators were connected to a microprocessor system. The virtual environment was created with the aid of a PC running a 3D virtual engine.

Given the details of the hardware and software design of the tactile glove, as well as the information associated with latency, it was possible to note that the results were quite significant because, with the architecture used, the total latency of the tactile internet environment was within the limits of restriction. Similar results were not found in the literature, which means that the proposed model is unprecedented. It was possible to verify the feasibility of using microprocessor systems to perform the control of the devices since the limits obtained with the proposed model, the latency of the devices were below the restriction limits.

It can be concluded that it is possible to use hardware models that use both the reconfigurable architecture and the microprocessor architecture to perform control of haptic devices in environments that require little latency for the devices, such as the tactile internet. As shown in the results, using reconfigurable FPGA computing it was possible to leave the total latency of the environment far below 1 ms and using microprocessed systems, the total latency associated with the environment was within the 10 ms limit, that is, the implementations of hardware were valid and fulfilled the objective because they meet the necessary prerequisites to operate in a tactile environment of the Internet. Given these characteristics, hardware models can be used to meet the future demand for tactile devices that have greater processing and can be used for commercial applications. This will be possible through the development of an IP core containing the implementations that have been presented.

4.1 Future works

A proposal for future work involves studies of the tactile environment that has been proposed, in order to further improve the round-trip latency of the system components. One point to be studied and analyzed is the impact it would have on the total latency of the environment when using predictive and adaptive control schemes in conjunction with the tactile system. It is possible that the use of forecast modules will result in some positive effect on system performance when the latency of the environment is high. Another valid additional contribution that can improve the latency of the environment is to use a better technique for closed-loop control of devices, currently, the traditional technique based on PID is used, if more innovative techniques based on artificial intelligence are used the process can be improved.

Another way for future work would be to analyze and implement improvements in the designs of the hardware models in order to further improve their performance. In the case of strategies that use reconfigurable computing, one of the improvements would be to optimize the CORDIC component that is responsible for calculating trigonometric functions. Along the same way, the modules that do the square root calculation and the

module that performs the division could also be optimized, because they are implemented only in floating-point.

Another proposition is in relation to the strategy that uses the microprocessor system, in the case of the tactile glove, it would be interesting that more IMU sensors were added to make it possible to capture the movement of the arm, and add more vibrotactile actuators so that it was possible to pass more reality in sensing. Within the same context, but in the part of the virtual environment, adding more types of textures and other types of materials can improve the usability of the system. Another valid additional contribution to the work is to develop other high-speed hardware modules that are associated with other types of haptic devices and also to analyze other possibilities of applications to increase the usability of the tactile environment.

Bibliography

Aijaz, A. (2016), Towards 5g-enabled tactile internet: Radio resource allocation for haptic communications, *em* '2016 IEEE Wireless Communications and Networking Conference', pp. 1–6.

Aijaz, A., M. Dohler, A. H. Aghvami, V. Friderikos & M. Frodigh (2016), 'Realizing the tactile internet: Haptic communications over next generation cellular networks', *IEEE Wireless Communications* **PP**(99), 2–9.

Aijaz, Adnan (2016), Towards 5g-enabled tactile internet: Radio resource allocation for haptic communications, *em* '2016 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)', IEEE, pp. 145–150.

Aijaz, Adnan, Mischa Dohler, A. Hamid Aghvami, Vasilis Friderikos & Magnus Frodigh (2015), 'Realizing the tactile internet: Haptic communications over next generation 5g cellular networks', *CoRR* **abs/1510.02826**.
URL: *http://arxiv.org/abs/1510.02826*

Antonakoglou, K., X. Xu, E. Steinbach, T. Mahmoodi & M. Dohler (2018*a*), 'Toward haptic communications over the 5g tactile internet', *IEEE Communications Surveys Tutorials* **20**(4), 3034–3059.

Antonakoglou, K., X. Xu, E. Steinbach, T. Mahmoodi & M. Dohler (2018*b*), 'Toward haptic communications over the 5g tactile internet', *IEEE Communications Surveys Tutorials* **20**(4), 3034–3059.

Aristidou, A. & J. Lasenby (2010), Motion capture with constrained inverse kinematics for real-time hand tracking, *em* 'Communications, Control and Signal Processing (ISCCSP), 2010 4th International Symposium on', pp. 1–5.

Arjun, N, SM Ashwin, Kurian Polachan, TV Prabhakar & Chandramani Singh (2018), An end to end tactile cyber physical system design, *em* '2018 4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems (EITEC)', IEEE, pp. 9–16.

Ben-Tzvi, Pinhas & Zhou Ma (2014), 'Sensing and force-feedback exoskeleton (safe) robotic glove', *IEEE Transactions on Neural Systems and Rehabilitation Engineering* **23**(6), 992–1002.

Berg, D. Van Den, R. Glans, D. De Koning, F. A. Kuipers, J. Lugtenburg, K. Polachan, P. T. Venkata, C. Singh, B. Turkovic & B. Van Wijk (2017), 'Challenges in haptic communications over the tactile internet', *IEEE Access* **5**, 23502–23518.

Bócsi, B., D. Nguyen-Tuong, L. Csató, B. Schölkopf & J. Peters (2011), Learning inverse kinematics with structured prediction, *em* '2011 IEEE/RSJ International Conference on Intelligent Robots and Systems', pp. 698–703.

Cavusoglu, Murat Cenk & David Feygin (2001), 'Kinematics and dynamics of phantom (tm) model 1.5 haptic interface'.

Chen, Youdong & Ling Li (2018), 'Predictable trajectory planning of industrial robots with constraints', *Applied Sciences* **8**(12).
URL: *https://www.mdpi.com/2076-3417/8/12/2648*

Coutinho, M. G. F., M. F. Torquato & M. A. C. Fernandes (2019), 'Deep neural network hardware implementation based on stacked sparse autoencoder', *IEEE Access* **7**, 40674–40694.

Culbertson, Heather, Samuel B Schorr & Allison M Okamura (2018), 'Haptics: The present and future of artificial touch sensation', *Annual Review of Control, Robotics, and Autonomous Systems* **1**, 385–409.

Culjat, M. O., J. Son, R. E. Fan, C. Wottawa, J. W. Bisley, W. S. Grundfest & E. P. Dutson (2010), Remote tactile sensing glove-based system, *em* '2010 Annual International Conference of the IEEE Engineering in Medicine and Biology', pp. 1550–1554.

Da Costa, A. L. X., C. A. D. Silva, M. F. Torquato & M. A. C. Fernandes (2019), 'Parallel implementation of particle swarm optimization on fpga', *IEEE Transactions on Circuits and Systems II: Express Briefs* pp. 1–1.

Da Silva, L. M. D., M. F. Torquato & M. A. C. Fernandes (2019), 'Parallel implementation of reinforcement learning q-learning technique for fpga', *IEEE Access* **7**, 2782–2798.

Dahiya, Ravinder (2019), 'E-skin: From humanoids to humans', *Proceedings of the IEEE* **107**(2), 247–252.

Dahiya, Ravinder S, Giorgio Metta, Maurizio Valle & Giulio Sandini (2010), 'Tactile sensing-from humans to humanoids.', *IEEE Trans. Robotics* **26**(1), 1–20.

de Souza, Alisson CD & Marcelo AC Fernandes (2014), 'Parallel fixed point implementation of a radial basis function network in an fpga', *Sensors* **14**(10), 18223–18243.

Dohler, M. (2015), The tactile internet iot, 5g and cloud on steroids, *em* '5G Radio Technology Seminar. Exploring Technical Challenges in the Emerging 5G Ecosystem', pp. 1–16.

Dohler, M., T. Mahmoodi, M. A. Lema, M. Condoluci, F. Sardis, K. Antonakoglou & H. Aghvami (2017), Internet of skills, where robotics meets ai, 5g and the tactile internet, *em* '2017 European Conference on Networks and Communications (EuCNC)', pp. 1–5.

Dohler, Mischa & G Fettweis (2015), *The tactile Internet-IoT, 5G and cloud on steroids*, IET.

D'Abbraccio, Jessica, Luca Massari, Sahana Prasanna, Laura Baldini, Francesca Sorgini, Giuseppe Airò Farulla, Andrea Bulletti, Marina Mazzoni, Lorenzo Capineri, Arianna Menciassi et al. (2019), 'Haptic glove and platform with gestural control for neuromorphic tactile sensory feedback in medical telepresence', *Sensors* **19**(3), 641.

Fang, Honggen, Zongwu Xie & Hong Liu (2009), An exoskeleton master hand for controlling dlr/hit hand, *em* '2009 IEEE/RSJ International Conference on Intelligent Robots and Systems', IEEE, pp. 3703–3708.

Feng, Di, Mohsen Kaboli & Gordon Cheng (2018), 'Active prior tactile knowledge transfer for learning tactual properties of new objects', *Sensors* **18**(2), 634.

Fettweis, G. P. (2014), 'The tactile internet: Applications and challenges', *IEEE Vehicular Technology Magazine* **9**(1), 64–70.

Fettweis, G.P. (2014), 'The tactile internet: Applications and challenges', *Vehicular Technology Magazine, IEEE* **9**(1), 64–70.

Franc, Marko & Aleš Hace (2013), 'A study on the fpga implementation of the bilateral control algorithm towards haptic teleoperation', *Automatika–Journal for Control, Measurement, Electronics, Computing and Communications* **54**(1).

Gac, K., G. Karpiel & M. Petko (2012), Fpga based hardware accelerator for calculations of the parallel robot inverse kinematics, *em* 'Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)', pp. 1–4.

Geomagic (n.d.), *Phantom Omni, Device Guide*.

Jiang, Z., Y. Dai, J. Zhang & S. He (2017), Kinematics calculation of minimally invasive surgical robot based on fpga, *em* '2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)', pp. 1726–1730.

Junior, José C. V. S., Matheus F. Torquato, Daniel H. Noronha, Sérgio N. Silva & Marcelo A. C. Fernandes (2019), 'Proposal of the tactile glove device', *Sensors* **19**(22).
URL: *https://www.mdpi.com/1424-8220/19/22/5029*

Kaboli, Mohsen, Di Feng & Gordon Cheng (2018), 'Active tactile transfer learning for object discrimination in an unstructured environment using multimodal robotic skin', *International Journal of Humanoid Robotics* **15**(01), 1850001.

Kaboli, Mohsen & Gordon Cheng (2018), 'Robust tactile descriptors for discriminating objects from textural properties via artificial robotic skin', *IEEE Transactions on Robotics* **34**(4), 985–1003.

Kaboli, Mohsen, Kunpeng Yao, Di Feng & Gordon Cheng (2019), 'Tactile-based active object discrimination and target object search in an unknown workspace', *Autonomous Robots* **43**(1), 123–152.

Kaboli, Mohsen, Rich Walker, Gordon Cheng et al. (2015), In-hand object recognition via texture properties with robotic hands, artificial skin, and novel tactile descriptors, *em* '2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)', IEEE, pp. 1155–1160.

Kumar, A., P. J. Gaidhane & V. Kumar (2017), A nonlinear fractional order pid controller applied to redundant robot manipulator, *em* '2017 6th International Conference on Computer Applications In Electrical Engineering-Recent Advances (CERA)', pp. 527–532.

Li, Chong, Chih-Ping Li, Kianoush Hosseini, Soo Bum Lee, Jing Jiang, Wanshi Chen, Gavin Horn, Tingfang Ji, John E Smee & Junyi Li (2018), '5g-based systems design for tactile internet', *Proceedings of the IEEE* **107**(2), 307–324.

Lin, Bor-Shing, I Lee, Shu-Yu Yang, Yi-Chiang Lo, Junghsi Lee, Jean-Lon Chen et al. (2018), 'Design of an inertial-sensor-based data glove for hand function evaluation', *Sensors* **18**(5), 1545.

Linh, Hai, Bui Thi & Ying-Shieh Kung (2015), 'Digital hardware realization of forward and inverse kinematics for a five-axis articulated robot arm', *Mathematical Problems in Engineering* **2015**.

Lobo, Jorge & Pedro Trindade (2013), 'Inertouchhand system-ith-demonstration of a glove device with distributed inertial sensors and vibro-tactile feedback', *International Journal of Online Engineering (iJOE)* **9**(S8), 56–58.

Lopes, Felipe F., João Canas Ferreira & Marcelo A. C. Fernandes (2019), 'Parallel implementation on fpga of support vector machines using stochastic gradient descent', *Electronics* **8**(6).

Maier, Martin, Mahfuzulhoq Chowdhury, Bhaskar Prasad Rimal & Dung Pham Van (2016), 'The tactile internet: vision, recent progress, and open challenges', *IEEE Communications Magazine* **54**(5), 138–145.

McCaw, John, Michelle C Yuen & Rebecca Kramer-Bottiglio (2018), Sensory glove for dynamic hand proprioception and tactile sensing, *em* 'ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference', American Society of Mechanical Engineers Digital Collection.

Mohammadi, Alireza, Jim Lavranos, Peter Choong & Denny Oetomo (2018), Flexoglove: A 3d printed soft exoskeleton robotic glove for impaired hand rehabilitation and assistance, *em* '2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)', IEEE, pp. 2120–2123.

Moskvitch, Katia (2015), 'Tactile internet: 5g and the cloud on steroids', *Engineering Technology* **10**(4), 48–53.

Muramatsu, Yuichi, Mihoko Niitsuma & Trygve Thomessen (2012), Perception of tactile sensation using vibrotactile glove interface, *em* '2012 IEEE 3rd International Conference on Cognitive Infocommunications (CogInfoCom)', IEEE, pp. 621–626.

N, A., A. S M, K. Polachan, P. T V & C. Singh (2018), An end to end tactile cyber physical system design, *em* '2018 4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems (EITEC)', pp. 9–16.

Nasrallah, Ahmed, Akhilesh S Thyagaturu, Ziyad Alharbi, Cuixiang Wang, Xing Shao, Martin Reisslein & Hesham ElBakoury (2018), 'Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research', *IEEE Communications Surveys & Tutorials* **21**(1), 88–145.

Noronha, Daniel H., Matheus F. Torquato & Marcelo A.C. Fernandes (2019), 'A parallel implementation of sequential minimal optimization on fpga', *Microprocessors and Microsystems* **69**, 138 – 151.

Oballe-Peinado, O., J. Castellanos-Ramos, J. A. Hidalgo, F. Vidal-Verdu, H. Macicior & E. Ochoteco (2009), Interface for tactile sensors based on direct connection to a fpga, *em* 'Mechatronics, 2009. ICM 2009. IEEE International Conference on', pp. 1–6.

Ohnishi, K. (2010), Real world haptics and telehaptics for medical applications, *em* '2010 IEEE International Symposium on Industrial Electronics', pp. 11–14.

O'Malley, Marcia K, Kevin S Sevcik & Emilie Kopp (2009), 'Improved haptic fidelity via reduced sampling period with an fpga-based real-time hardware platform', *Journal of Computing and Information Science in Engineering* **9**(1), 011002.

Popescu, Nirvana, Dorin Popescu, Marian Poboroniuc & Cristian Dinu Popescu (2013), Intelligent haptic robotic glove for patients diagnosed with cerebrovascular accidents, *em* '2013 17th International Conference on System Theory, Control and Computing (ICSTCC)', IEEE, pp. 717–721.

Rahimi, H.N. & M. Nazemizadeh (2014), 'Dynamic analysis and intelligent control techniques for flexible manipulators: a review', *Advanced Robotics* **28**(2), 63–76.

Sagisaka, Takashi, Yoshiyuki Ohmura, Akihiko Nagakubo, Kazuyuki Ozaki & Yasuo Kuniyoshi (2012), Development and applications of high-density tactile sensing glove, *em* 'International Conference on Human Haptic Sensing and Touch Enabled Computer Applications', Springer, pp. 445–456.

Sagisaka, Takashi, Yoshiyuki Ohmura, Yasuo Kuniyoshi, Akihiko Nagakubo & Kazuyuki Ozaki (2011), High-density conformable tactile sensing glove, *em* '2011 11th IEEE-RAS International Conference on Humanoid Robots', IEEE, pp. 537–542.

San Martin, Jose & Gracián Triviño (2006), A study of the manipulability of the phantom omni haptic interface., *em* 'VRIPHYS', pp. 127–128.

Sánchez, Diego F, Daniel M Muñoz, Carlos H Llanos & José M Motta (2010), 'A reconfigurable system approach to the direct kinematics of a 5 dof robotic manipulator', *International Journal of Reconfigurable Computing* **2010**.

Sano, Yuko, Naoki Wake, Akimichi Ichinose, Michihiro Osumi, Reishi Oya, Masahiko Sumitani, Shin-ichiro Kumagaya & Yasuo Kuniyoshi (2016), 'Tactile feedback for relief of deafferentation pain using virtual reality system: a pilot study', *Journal of neuroengineering and rehabilitation* **13**(1), 61.

Sansanayuth, T., I. Nilkhamhang & K. Tungpimolrat (2012), Teleoperation with inverse dynamics control for phantom omni haptic device, *em* '2012 Proceedings of SICE Annual Conference (SICE)', pp. 2121–2126.

Shen, Shaobo, Aiguo Song & Tao Li (2019), 'Predictor-based motion tracking control for cloud robotic systems with delayed measurements', *Electronics* **8**(4).

Shirafuji, Shouhei & Koh Hosoda (2014), 'Detection and prevention of slip using sensors with different properties embedded in elastic artificial skin on the basis of previous experience', *Robotics and Autonomous Systems* **62**(1), 46–52.

Silva, Alejandro Jarillo, Omar A Domínguez Ramirez, Vicente Parra Vega & Jesus P Ordaz Oliver (2009), Phantom omni haptic device: Kinematic and manipulability, *em* 'Electronics, Robotics and Automotive Mechanics Conference, 2009. CERMA'09.', IEEE, pp. 193–198.

Simsek, M., A. Aijaz, M. Dohler, J. Sachs & G. Fettweis (2016*a*), The 5g-enabled tactile internet: Applications, requirements, and architecture, *em* '2016 IEEE Wireless Communications and Networking Conference', pp. 1–6.

Simsek, M., A. Aijaz, M. Dohler, J. Sachs & G. Fettweis (2016*b*), The 5g-enabled tactile internet: Applications, requirements, and architecture, *em* '2016 IEEE Wireless Communications and Networking Conference', pp. 1–6.

Simsek, Meryem, Adnan Aijaz, Mischa Dohler, Joachim Sachs & Gerhard Fettweis (2016*c*), '5g-enabled tactile internet', *IEEE Journal on Selected Areas in Communications* **34**(3), 460–473.

Song, G., S. Guo & Q. Wang (2006), A tele-operation system based on haptic feedback, *em* '2006 IEEE International Conference on Information Acquisition', pp. 1127–1131.

Szabo, D., A. Gulyas, F. H. P. Fitzek & D. E. Lucani (2015), Towards the tactile internet: Decreasing communication latency with network coding and software defined networking, *em* 'European Wireless 2015; 21th European Wireless Conference; Proceedings of', pp. 1–6.

Szabo, David, Andras Gulyas, Frank H.P. Fitzek, Frank H.P. Fitzek & Daniel E. Lucani (2015), Towards the tactile internet: Decreasing communication latency with network coding and software defined networking, *em* 'European Wireless 2015; 21th European Wireless Conference; Proceedings of', pp. 1–6.

Tanaka, Hiroyuki, Kouhei Ohnishi & Hiroaki Nishi (2009), Haptic communication system using fpga and real-time network framework, *em* 'Industrial Electronics, 2009. IECON'09. 35th Annual Conference of IEEE', IEEE, pp. 2931–2936.

Tang, Sai Hong, Chun Kit Ang, Mohd Khairol Anuar Bin Mohd Ariffin & Syamsiah Binti Mashohor (2014), 'Predicting the motion of a robot manipulator with unknown trajectories based on an artificial neural network', *International Journal of Advanced Robotic Systems* **11**(10), 176.

Torquato, Matheus F. & Marcelo A. C. Fernandes (2019), 'High-performance parallel implementation of genetic algorithm on fpga', *Circuits, Systems, and Signal Processing* .

Ulmen, John & Mark Cutkosky (2010), A robust, low-cost and low-noise artificial skin for human-friendly robots, *em* '2010 IEEE International Conference on Robotics and Automation', IEEE, pp. 4836–4841.

Union, International Telecommunication (2014), 'The tactile internet itu-t technology watch report'.
URL: *https://www.itu.int/oth/T2301000023/en*

Volder, Jack E (1959), 'The cordic trigonometric computing technique', *IRE Transactions on Electronic Computers* (3), 330–334.

Weber, Paul, E Rueckert, Roberto Calandra, Jan Peters & Philipp Beckerle (2016), A low-cost sensor glove with vibrotactile feedback and multiple finger joint and hand motion sensing for human-robot interaction, *em* '2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)', IEEE, pp. 99–104.

Wong, C. C. & C. C. Liu (2013), 'Fpga realisation of inverse kinematics for biped robot based on cordic', *Electronics Letters* **49**(5), 332–334.

Wu, M., Y. Kung, Y. Huang & T. Jung (2014), Fixed-point computation of robot kinematics in fpga, *em* '2014 International Conference on Advanced Robotics and Intelligent Systems (ARIS)', pp. 35–40.

Xiang, Yu (2019), 'Simulation and analysis of three-dimensional space path prediction for six-degree-of-freedom (sdof) manipulator', *3D Research* **10**(2), 15.

Yang, Chenguang, Hongbin Ma & Mengyin Fu (2016), *Intelligent Control of Robot Manipulator*, Springer Singapore, Singapore, pp. 49–96.

Yang, Chongjun, Yu Xie, Shuang Liu & Dong Sun (2018*a*), Force modeling, identification, and feedback control of robot-assisted needle insertion: A survey of the literature, *em* 'Sensors'.

Yang, Chongjun, Yu Xie, Shuang Liu & Dong Sun (2018*b*), 'Force modeling, identification, and feedback control of robot-assisted needle insertion: a survey of the literature', *Sensors* **18**(2), 561.

Yang, Dongseok & Younggeun Choi (2018), Palm glove: wearable glove based on palm-camera for thumb-to-finger tap recognition, *em* '2018 International Conference on Information and Communication Technology Convergence (ICTC)', IEEE, pp. 549–551.

Yao, Kunpeng, Mohsen Kaboli & Gordon Cheng (2017), Tactile-based object center of mass exploration and discrimination, *em* '2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)', IEEE, pp. 876–881.

Yogeswaran, Nivasan, Wenting Dang, William Taube Navaraj, Dhayalan Shakthivel, Saleem Khan, Emre Ozan Polat, Shoubhik Gupta, Hadi Heidari, Mohsen Kaboli, Leandro Lorenzelli et al. (2015), 'New materials and advances in making electronic skin for interactive robots', *Advanced Robotics* **29**(21), 1359–1373.

Yu, Q., C. Wang, X. Ma, X. Li & X. Zhou (2015), A deep learning prediction process accelerator based fpga, *em* '2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing', pp. 1159–1162.