A GESTURE DETECTION GLOVE FOR HUMAN-COMPUTER INTERACTION

Felipe A. Quirino¹, Marcelo Romanssini², Rafael R. Dorneles³, Enzo H. Weber³ and Alessandro Girardi⁴

¹ Computer Science undergraduate student, Federal University of Pampa, Alegrete - RS, Brazil
² Electrical Engineering undergraduate student, Federal University of Pampa, Alegrete - RS, Brazil
³ High school student, Divino Coração School, Alegrete - RS, Brazil
⁴ Advisor, Federal University of Pampa, Alegrete - RS, Brazil

ABSTRACT

The use of portable electronic devices, such as smartphones and tablets, is diffused around the world. However, because of the reduced size, the interaction with the human being is difficult, especially the data input by typing on virtual keyboards in the screen. Also, it is necessary to develop new interfaces that allow a more agile human-computer interaction with less effort to the user. This report describes the development of a gesture capture glove capable of recognizing patterns of hand movement and transform them into useful information. The proposed system consists of six accelerometers and six gyroscopes located at the fingertips and over the posterior part of the hand, allowing precise detection of movements in three-dimensional space. A series of applications were developed, such as the recognition of gesture patterns of sign languages, allowing people with some kind of special needs (blind or deaf-mute, for example) to interact efficiently with a computer or smartphone. The glove may replace conventional joysticks and mice devices with the advantage of inserting more degrees of freedom to the control. We also developed a 3D virtual hand model controlled by the glove. The proposed system is cheap and can be implemented with conventional electronics devices available in the market.

1. INTRODUCTION

The use of portable devices, such as smartphones, smartwatches and tablets, is becoming increasingly widespread among ordinary users. These devices make people’s everyday lives easier, and they are fundamental in the development of new applications [1]. However, because of the reduced size, the interaction with the human being is difficult, especially the data input by typing on virtual keyboards in the screen. Writing a reasonably sized text in these devices is uncomfortable and can cause physical damage to the body - such as RSI (repetitive strain injury) and tendonitis - if used by a long period. Thus, it is necessary to develop new interfaces that allow a more agile human-computer interaction with less effort to the user.

This project aims to develop a gesture detection glove capable of recognizing patterns of hand movement and transform them into useful information.

The proposed system consists of six accelerometers and six gyroscopes [3] located at the fingertips and over the posterior part of the hand, allowing precise detection of movements in three-dimensional space. The sensors are read by a microcontroller, which processes the values of linear and angular acceleration and sends the information to the final device (a computer, a smartphone or a tablet).

The developed glove has a lot of applications, such as the recognition of movement patterns of sign languages for writing on digital equipment [2]. It allows people with some kind of special needs (blind or deaf-mute, for example) to interact more efficiently with a computer. The glove may replace conventional joysticks and mice devices with the advantage of inserting more degrees of freedom to the control. Another uses can also be cited, such as digital game control, virtual reality interface, motion detection for physiotherapy exercises, training for health academics and remote control of surgeries. Also, the glove may be used as a platform for teaching activities through the development of applications that can exploit its functionalities.
The development of gesture capture devices has been reported previously in the literature. The Smart Glove project [2] is the one that most closely resembles the project proposed here. It uses both three-axes accelerometers and pressure sensors to detect the curves between the fingers and capture the movements of the hand. When the user makes gesture equivalent to a word in sign language, the computer emits a sound equivalent to the detected word. To circumvent the problem that different people make different gestures, a machine learning technique is used. It is able to identify which gestures belong to a particular group from the analysis of several samples.

The hand gesture can also be detected indirectly by the sound emitted by the user pulse. The design described by [5] describes a device based on this data. The pulse emits an specific sound wave for each movement, which can be captured by a sensor. The gesture recognition process is similar to the previously mentioned project. This device is inexpensive because it uses only five microphones around the wrist. However, some gestures are difficult to recognize, because some different hand movements emit similar sound waves.

Kinect is another motion recognition device very used for body movement detection [12]. It consists of a camera that capture three-dimensional images in conjunction with an infrared projector and a monochrome CMOS sensor to create the 3D environment and detect its modifications. A large and enclosed space is necessary for its operation. The equipment eliminates the use of any type of device placed at the human body, facilitating the mobility of the user. The camera with depth allows a better capture of images, which facilitates the discovery of the position of the user as he moves. The drawback is that the user needs to stay in front of the camera viewing area, reducing the portability of the application.

The Leap Motion [11] is another interesting recognition device. It recognizes fingers and hand movements by means of infrared LEDs. This equipment does not need any contact with the hand and it is portable. The sensors must be positioned close to the hands to provide a good accuracy.

However, it is possible to verify that the gesture capturing by means of accelerometers and gyroscopes presents advantages in relation to other approaches, mainly because of the robustness. There is no interference according to the luminosity, skin color or anatomy from different users. In addition, the ease of use, intuitive operation and freedom of movement in any environment makes a glove a good alternative for automatic hand movement recognition.

This report describes the development of a hand gesture recognition glove based on accelerometers and gyroscopes capable to identify and interpret humans gestures. The remaining sections are organized as following: Section 2 presents the details about the system implementation, Section 3 describes some applications using the glove and Section 4 presents the concluding remarks.

2. SYSTEM IMPLEMENTATION

The relationship between man and machine is becoming more natural and less complex with the evolution of the technology. However, the advances of portable devices, such as smartphones, smartwatches and tablets, have turned it difficult for human beings to interact with small input keyboards. The implementation of the gesture capture glove has the objective of improving the interaction with these devices, since, through gestures, the user can transmit the intention of performing a certain activity or command.

![Fig. 1: General scheme of the gesture capture glove system.](image-url)
Fig. 1 shows a general scheme of the proposed system. The microcontroller reads the hand movement data provided by sensors and sends them to the final device via Bluetooth protocol. The final device can be a computer, smartphone or tablet.

Six accelerometers and gyroscopes capture hand gestures by linear and angular acceleration data for each finger and for the dorsal area of the hand. The PAMPIUM microcontroller is used to control the glove, performing configuration and capturing sensor data. Fig. 2 shows the detailed block diagram of the entire system. The communication between the sensors and PAMPIUM is made by the I2C communication protocol. The communication between the PAMPIUM and the Bluetooth interface (HC05) is implemented by the RS-232 communication protocol. Finally, the communication with the final device happens through the Bluetooth protocol, which does not require any wires.

The sensors are of type MPU 6050 and are placed at the end of each finger and at the back of the hand, thus allowing a good sensitivity to the movements. The I2C and RS-232 communication interfaces are implemented in FPGA along with the PAMPIUM microcontroller. For this, we used in a small FPGA 101 development board placed at the back of the hand.

Since six MPU 6050 sensors are used, it is necessary to use a multiplexer/demultiplexer unit for communication between them and the microcontroller. This allows only a single I2C communication channel to be implemented in PAMPIUM.

Because all PAMPIUM operations happen between registers, we connect the communication interfaces directly to the register bank, making it easier to implement control protocols for both sending and receiving data.

Fig. 3 shows the location of the sensors on the glove. Each MPU 6050 has a three-axes accelerometer, responsible for measuring the linear acceleration on the x, y and z-axes, as well as a three-axes gyroscope that measures the angular acceleration over the x, y and z-axes, thus obtaining a good notion of three-dimensional space. The accelerometer is capable to measure accelerations in the three axes separately up to 16g (156.896 m/s²) and has four programmable ranges: 2g, 4g, 6g and 16g. The unit g refers to the value of the Earth gravitational acceleration. The gyroscope is capable to measure instantaneous variations on the three axes separately with four
programmable ranges: 250º/s, 500º/s, 1000º/s and 2000º/s [3].

Communication through the I2C standard occurs at a maximum rate of 400 kHz and operates in a supply voltage range from 2.37 V to 3.46 V. The voltage levels for the communication signals are identical to the values of power supply, i.e., 0 V for logic level ‘0’ and 2.37 V to 3.46 V for logic level ‘1’.

As can be seen in Fig. 4, the MPU 6050 is commercially available in a module containing eight pins, in which VCC and GND are power supply pins and SCL and SDA are communication pins that follow the I2C protocol. Through the AD0 pin it is possible to change the access address of the device, allowing two sensors being controlled by the same master. The MPU 6050 can also function as master, controlling an external sensor, which can be of pressure, direction, magnetic flow, among others. The XDA, SCI and INT pins are related to the external control [3] but are not used in this project.

The MPU 6050 digitizes the x, y and z-axis values collected from the gyroscopes and accelerometers by means of six A/D (analog/digital) converters - three A/D for the accelerometer and three A/D for the gyroscope. The read values are stored in internal registers. The size of each internal register is 8 bits, and the precision of the gyroscopes and accelerometers is 16 bits. Therefore, each collected value is stored in two internal registers, one for the 8 most significant bits and the other for the 8 least significant ones. The values are stored in two’s complement format.

The HC05 is a Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial communication, which makes it easy to interface with the final device. Fig. 5 shows the HC05 Bluetooth module used in this project.

Fig. 3: Location of the six MPU 6050 sensors on the glove.

Fig. 4: MPU 6050 module.

Fig. 5: HC05 Bluetooth module.
2.1 PAMPIUM Microcontroller

PAMPIUM is a fully configurable microcontroller designed by our research group of Federal University of Pampa (UNIPAMPA). This microcontroller has a general purpose 16-bit RISC architecture described in System Verilog. The circuit description is free and open source [4]. Because it is configurable, it is possible to scale all the internal modules according to the application needs, which makes it as efficient as possible in terms of power consumption, area and performance. PAMPIUM is available in monocyce, multicycle, pipeline and superscalar versions.

The internal organization of PAMPIUM consists of two memories - one of program and one of instructions - two register banks, one arithmetic-logic unit (ALU) and one control unit. Fig. 6 illustrates the microcontroller basic internal organization.

![Fig. 6: Basic organization of PAMPIUM microcontroller.](image)

The I/O interface module can be specified for any communication protocol. Specific bits contained in the register bank define the directionality of each communication pin. The data reading and writing on the ports are also stored in dedicated registers [4].

The PAMPIUM register bank consists of two secondary banks called R0 and R1. The number and the size of each register are configurable and defined by the user. In our project, 32 registers were used for each bank and each register has a word length of 16 bits. In order to access a certain register it is necessary to indicate its address. For example, to access register 12 in bank R0 we write [#12]R0 in assembly notation. Some registers have specific functionalities for instructions or port configurations and others are for general purpose. Fig. 7 shows the basic scheme of the register bank.

![Register Bank R0 and R1](image)

Fig. 7: Basic scheme of the PAMPIUM register bank.

The program memory in the implemented version of PAMPIUM microcontroller is composed of a memory block with a 24-bit word size and a 16-bit program counter. The memory block has the capacity to store a maximum of 64 kWord instructions, which is enough for our application. The program counter has the function of indicating which instruction is being executed and the calculation of the next instruction address. Fig. 8 shows the scheme of program memory and program counter.

![Program Memory and Program Counter](image)

Fig. 8: Scheme of program memory and program counter in the PAMPIUM microcontroller.

The 6 most significant bits of the instruction word indicate the operation code. The next 2 bits indicate in which bank are located the registers used in the
operation. Finally, the remaining 16 bits are used according to the instruction type. Fig. 9 shows the basic instruction format.

![Fig. 9: Basic instruction format.](image)

The instructions are classified into three formats, called L, R and M.

The L format represents the operations that require literal values to execute. We divide this format into four distinct fields, as shown in Fig. 10: a field for the definition of the operation code, two fields A and B that inform at which register bank is located the register used by the operations and a field of 16 bits for the entry of literals.

![Fig. 10: L instruction format.](image)

The R format is used by instructions that perform operations between registers. We divide this format into seven distinct fields as shown in Fig. 11: a field for the definition of the operation code, three fields A, B and C that inform the registers used by the operation and three fields RA, RB and RC that define the address of these registers.

![Fig. 11: R instruction format.](image)

The M format is used by the instructions that perform operations between a register and the data memory. We divide this format into six distinct fields as shown in Fig. 12: a field for the definition of the operation code, three fields A, B and C that inform at which register bank are located the registers used by the operation, the RA field, which defines the address of the register being used in the operation, and the BIAS field, which is used to calculate the instruction memory access address.

![Fig. 12: M instruction format.](image)

The instructions supported by the PAMPIUM microcontroller can be divided in three subsets: basic, operative and data conversion. The basic subset is independent of the data type and contains 13 instructions, as shown in Tab. 1.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>No operation</td>
</tr>
<tr>
<td>END</td>
<td>Freeze the program</td>
</tr>
<tr>
<td>JUMP</td>
<td>Jump</td>
</tr>
<tr>
<td>CALL</td>
<td>Jump and save return address</td>
</tr>
<tr>
<td>RET</td>
<td>Function return</td>
</tr>
<tr>
<td>RETL</td>
<td>Function return saving literal</td>
</tr>
<tr>
<td>BBCLEAR</td>
<td>Branch if bit equal to &quot;0&quot;</td>
</tr>
<tr>
<td>BBSET</td>
<td>Branch if bit equal to &quot;1&quot;</td>
</tr>
<tr>
<td>SHR</td>
<td>Shift right</td>
</tr>
<tr>
<td>SHL</td>
<td>Shift left</td>
</tr>
<tr>
<td>BSET</td>
<td>Set a bit from a register</td>
</tr>
<tr>
<td>BCLEAR</td>
<td>Clear a bit from a register</td>
</tr>
<tr>
<td>SETUP</td>
<td>Set data memory pointer</td>
</tr>
</tbody>
</table>

![Table 1: Basic instruction subset.](image)

The operative subset is presented in Tab. 2. It is composed of 20 instructions that can be used for different data types. The "*" character can be substituted according to the following data types: "F" for operands of type Float, "I" for Integer, "C" for Char and "D" for Double.

The instructions in the data conversion subset convert the data from one type to another, according to presented in Tab. 3. All the implemented instructions are described in details in [4].
Table 2: Operative instruction subset.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*COPY</td>
<td>Copy values between registers</td>
</tr>
<tr>
<td>*MOVL</td>
<td>Assign literal to #*REG_L</td>
</tr>
<tr>
<td>*RM</td>
<td>Read data memory</td>
</tr>
<tr>
<td>*WM</td>
<td>Write in data memory</td>
</tr>
<tr>
<td>*BL</td>
<td>Branch if larger</td>
</tr>
<tr>
<td>*BS</td>
<td>Branch if smaller</td>
</tr>
<tr>
<td>*BLE</td>
<td>Branch if large or equal</td>
</tr>
<tr>
<td>*BSE</td>
<td>Branch if small or equal</td>
</tr>
<tr>
<td>*BE</td>
<td>Branch if equal</td>
</tr>
<tr>
<td>*BNE</td>
<td>Branch if no equal</td>
</tr>
<tr>
<td>*MULT</td>
<td>Multiplication</td>
</tr>
<tr>
<td>*DIV</td>
<td>Division</td>
</tr>
<tr>
<td>*ADD</td>
<td>Sum</td>
</tr>
<tr>
<td>*SUB</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*COMP</td>
<td>Complement</td>
</tr>
<tr>
<td>REM</td>
<td>Division remainder</td>
</tr>
<tr>
<td>OR</td>
<td>Logic OR</td>
</tr>
<tr>
<td>AND</td>
<td>Logic AND</td>
</tr>
<tr>
<td>XOR</td>
<td>Logic XOR</td>
</tr>
<tr>
<td>NOT</td>
<td>Logic NOT</td>
</tr>
</tbody>
</table>

PAMPIUM also has a data memory, configured in this project with 64 kB. The arithmetic logic unit (ULA) performs arithmetic logic operations between registers, such as sum, subtraction, division and remainder. The logical operations are AND, OR, NOT and XOR.

For this application, we defined that the monocycle version of PAMPIUM is a more adequate one, since the performance requirements are not critical. This version occupies a small area and presents low power consumption. The synthesis in FPGA allows operation at a frequency of up to 50MHz in an Altera EP4CE6E22C8N device, which is suitable for the requirements of both I2C and RS-232 communication protocols. Fig. 13 shows the complete organization of monocycle version of PAMPIUM used in this project.

2.2 I2C Interface Implementation

The I2C (Inter-Integrated Circuit) interface is a two-wire serial communication protocol for connecting low-speed devices such as microcontrollers, converters, among others. The communication in the I2C interface happens between a master and slave, i.e., between a device that commands the bus (master) and a device that responds to the commands sent by the master (slave). Each slave must have a unique address. The bus consists of two wires with pull-up resistors, to ensure that they are always in high logic level when the bus is not commanded by any device. The communication between master and slave is performed through the SDA (serial data) pin, which is responsible for transmitting data, and SCL (serial clock), that is responsible for synchronization. Fig. 14 shows the master-slaves connection.

In this protocol, communication happens serially in 8-bit packets. First, the master sends the start condition (S) to the slave, followed by the slave address (7 bits) and a final bit indicating whether the next operation will be reading or writing. Soon after, the device which receives the information sends an acknowledgment bit (ACK). To terminate the communication between master and slave, the master sends the stop condition (P). Fig. 15 shows the communication stages.

Table 3: Data conversion instruction subset.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITOF</td>
<td>Convert integer to float</td>
</tr>
<tr>
<td>ITOD</td>
<td>Convert integer to double</td>
</tr>
<tr>
<td>FTOI</td>
<td>Convert float to integer</td>
</tr>
<tr>
<td>DTOI</td>
<td>Convert double to integer</td>
</tr>
<tr>
<td>FTOD</td>
<td>Convert float to double</td>
</tr>
<tr>
<td>DTOF</td>
<td>Convert double to float</td>
</tr>
</tbody>
</table>
The ACK bit indicates that the device has finished receiving the data and that it is ready to receive another packet or to close the communication. To send the ACK bit the SCL bus must give a pulse while the SDA bus is in low logic level. If there is an interruption in the communication between master and slave in the middle of a read routine, a bit NACK is going to be sent at the end of the transmission, after which the stop condition is performed. In order to ensure this to happen, the bus has pull-up resistors, which guarantees that whenever an error occurs during transmission the bus goes to high logic level so that the NACK condition can be effected. To send a NACK bit, the SCL bus must give a pulse while the SDA bus is in logic high. Fig. 16 shows the ACK and NACK conditions.

Data transmission on the SDA bus only happens when the SCL bus is at high logic level, in order to avoid wrong starting and stopping conditions.
We implemented the I2C communication interface so that its control is exercised directly by the PAMPUM registers. For this, we allocate some bits of a specific register and connected in the I2C interface. The communication interface has two input/output pins (SCL and SDA) connected to the glove sensors via a multiplexor/demultiplexor. Fig. 17 shows the control block of the I2C communication interface and the input/output pins used for configuration and communication.

The interface pins are described as following:

- **SCL** and **SDA** are the communication pins of the I2C interface that connect to the glove sensors;
- **DIV** is a configuration pin, responsible for the internal division of the received clock signal. Its purpose is to adjust the communication speed of the control block in relation to the slave block. There are 3 control bits, which results in 8 different divisions of the input clock, as shown in Tab. 4. The values can be changed directly in register 0 of register bank R1 ([#0]R1), bits 3 to 5;
- **Data_in** and **Data_out** are communication pins of the interface with the microcontroller. The communication is done in packets of 8 bits. The value is allocated in register 4 of register bank R0 ([#4]R0);
- **EN_in** and **EN_out** are pins responsible for enabling the I2C communication interface. These pins are allocated in register 0 of register bank R0 ([#0]R0), in bit 0. The interface is active when the logic level of this bit is “1”;
- **BUSY_in** and **BUSY_out** are control pins that indicate when to start a write or read routine. The control bit is allocated to register 0 of register bank R0 ([#0]R0), in bit 1;
- **R/W** is the pin that indicates whether the executed action is writing at or reading on the slave device. This pin is allocated in register 0 of register bank R0 ([#0]R0), bit 2;
- **S** is the pin that controls the start condition (S). For the start condition occur the logical level of this pin should be “1”. This pin is allocated in register 0 of register bank R0 ([#0]R0), bit 3;
- **ACK** is the pin that indicates if the slave device sent the acknowledgment action (ACK). This pin

![I2C interface control block](image_url)

**Fig. 16:** Acknowledge condition (ACK) and transmission abort (NACK).

**Fig. 17:** I2C interface control block.

**Table 4: I2C interface clock division**

<table>
<thead>
<tr>
<th>Input (DIV)</th>
<th>Division value</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>2</td>
</tr>
<tr>
<td>010</td>
<td>4</td>
</tr>
<tr>
<td>011</td>
<td>8</td>
</tr>
<tr>
<td>100</td>
<td>16</td>
</tr>
<tr>
<td>101</td>
<td>32</td>
</tr>
<tr>
<td>110</td>
<td>64</td>
</tr>
<tr>
<td>111</td>
<td>128</td>
</tr>
</tbody>
</table>
is allocated in register 0 of register bank R0 ([#0]R0), bit 4;

- **EN_WF** is the pin that enables the writing of the communication interface in the register bank.
- **EN_WD** is the pin that enables the writing of data coming from the interface communication in the register bank;
- **CLK** is the clock signal, which is already adjusted for the interface communication frequency;
- **RST** is the reset input pin, which clears all register data and resets the communication interface to its initial configuration;
- **F_INT** is a pin that indicates when the interface has finished executing a reading or writing routine and is ready to receive a new command. This pin is allocated in register 0 of register bank R1 ([#0]R1), bit 2.

For the connection between PAMPIUM and sensors, we use the external connection pins in the FPGA. We perform this directly, because the voltage levels generated by the FPGA101 development board are equal to the levels required by the MPU 6050 sensors, i.e., 0V for logic level “0” and 3.3V for logic level “1”.

Fig. 18 shows the operating flow of the I2C interface. The communication starts when the ENA signal goes to logical level “1”. Then, the interface sends the start condition (S) and enters in a loop waiting for BUSY_in to go to logical level “1”. Analyzing the R/W bit, which indicates whether the next action will be writing or reading, the control module takes action. If R/W=0, the interface performs a write routine and checks the recognition bit (ACK). If ACK=0, the stop condition (P) is executed. In the case of ACK = 1, then a second check is made, this time of signal S, which indicates whether a new start condition must be executed. If R/W=1, then the interface performs a read routine checking S. If S=1, the transmission end condition (NACK) is executed and then the stop condition. If S=0, the interface returns to the loop and waits for BUSY_in = 1 to perform a new action. This cycle keeps repeating until the routine of writing or reading data is completed.

### 2.3 RS-232 Interface Implementation

The RS-232 interface is a standard protocol for serial data communication that uses negative voltage signals for logic level “1” and positive for logic level “0”, ranging from ±5 V to ±25 V depending on the application. This protocol presents two separate RX and TX communication channels, which makes it possible to send and receive data in parallel. Fig. 19 shows the communication between devices through the RS-232 protocol.

The start (S) condition occurs when the bus logic signal change from logic high to low. Then, 8 bits of data are sent. The stop (P) condition occurs when the bus remains at high logic level. The transmission rate of the RS-232 protocol can be set to any value between 300 to 230,400 bits per second. Therefore, the two devices must operate at the same rate for the communication to operate efficiently.

---

**Fig. 18: Flowchart of the I2C interface.**
We implemented the RS232 interface in a similarly way to the I2C interface, in which all control pins are connected directly to the register bank. For this, we allocated some bits of the control interface to some dedicated registers in the register bank. For the communication between the interface and the Bluetooth module, we used two pins (RX and TX) according to Fig. 20.

The interface pins are described below:

- **CLK** is the clock signal used for the control the receiving and transmission speed;
- **RST** is the reset signal input pin, which clears all register data and resets the communication interface to the initial configuration;
- **Data_in** is the data transfer pin from the microcontroller to the communication interface. It is allocated in register 16 of register bank R0 ([$#16$]R0), bits 0 to 7;
- **Data_out** is the data transfer pin of the communication interface for the microcontroller. This pin is allocated in register 17 of register bank R0 ([$#16$]R0), in bits 0 to 7;
- **EN** is responsible for enabling the RS232 communication interface.
- **GO_IN** and **GO_OUT** are pins responsible for sending data commands. When activated, the interface sends the data stored in register 16 of register bank R0. This pin is allocated in register 0 of register bank R0 ([$#N0$]R0), bit 6;
- **EN_WD** is the pin that enables the writing of data coming from the communication interface in the register bank;
- **EN_WF** is the pin that enables the writing of the communication interface in the register bank;
- **F_INT** is a pin that indicates when the interface finishes executing a send or receive routine and is ready to execute a new command. This pin is allocated in register 0 of register bank R1 ([$#N0$]R1), bit 6.

Fig. 21 shows the operating flow of the RS-232 interface. The interface starts when the ENA signal is set to logic level “1”. The baud rate is automatically detected through the auto baud procedure. For this, the interface reads the input waiting for a specific byte from the device connected to it. This byte was defined as ‘10101010’ (character “U” in ASCII code). The interface calculates the number of clock cycles necessary to receive this byte. Dividing the number of
cycles by 8 gives the transmission speed. After adjusting the communication speed, a test routine starts by monitoring if the interface must receive or send the next data. First, the RX pin is checked. If RX=0, a reading routine is started. If RX=1, the control tests GO_IN. If GO_IN=1, the writing routine is executed. At the end of this routine, the GO_IN pin resets and the control comes back to the verification step.

2.4 Interface Software

We implemented an interface driver for reading and processing data by the computer (final device) in Matlab. This interface receives accelerometers and gyroscopes data from the glove through Bluetooth and processes the information according to the application. A pre-treatment of the data is necessary for noise suppression and rotation detection. We normalize the sensor data to a range of -10 to +10 for the accelerometers and from $-\pi$ to $+\pi$ for the gyroscopes.

The gyroscopes identify the angles of rotation of the sensors in relation to a reference. Thus, we need to correct the accelerometer values in the X, Y and Z local axes for the fixed x, y and z axes of the application.

The start position of the sensors is used as reference for the correction of the movement in relation to the values read from the sensors. The hand must start in the horizontal position in order to the driver identify the amount of adjust necessary for the exact location of the sensors in relation to the application reference. Therefore, if the sensor is tilted relative to the external reference, the acceleration must be adjusted for each axis.

Fig. 22 shows the intrinsic rotation of the sensor axes to the fixed coordinate system of the application space. The angles $\alpha$, $\beta$ and $\gamma$ represent the Euler angles that can be calculated with the values obtained by the gyroscopes.

To identify the coordinates of the vector in the application axes in a 3-dimensional space we use the following rotation matrixes:

$$R_x(y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(y) & -\sin(y) \\ 0 & \sin(y) & \cos(y) \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The multiplication of each rotation matrix produces the complete rotation of a given point.

$$R = R_z(\alpha)R_y(\beta)R_x(y)$$

Thus, we calculate the new coordinates $P'$ of a point $P$ as:

$$P' = R \cdot P$$

To eliminate noise and outlier points, we use a moving average low pass filter over the data read from
the sensors. In this filter, each point is replaced by the arithmetic mean of \( n \) points. The higher this value, the higher the degree of smoothing. In this design we use an arbitrary \( n = 11 \).

Fig. 23 demonstrates the flowchart of the implemented interface algorithm. A graphical interface implemented in Matlab, shown in Fig. 24, facilitates the visualization of the sensor data.

2.5 Prototype

The implementation of the gestural capture glove prototype encounters some challenges. The mobility of the user in the operation of the device is a necessity, since the free movement of the hand is a fundamental requirement for its correct operation. Thus, we try to minimize the amount of wires and the weight of the glove. However, since the current development state is a prototype, it can be further optimized in these aspects.

The 6 MPU6050 sensors were sewn over a conventional glove, as well as the FPGA 101 development board. This development board is one of the smallest size available in the market and fits well to the performance requirements. The operating frequency is 12.5 MHz, which is enough for communication with the sensors and with the Bluetooth module. Fig. 25 shows the portable version of the glove prototype. The power consumption of the entire system is estimated in 0.45W.

The list of materials used for implementing the prototype is depicted in Tab. 5. The total cost is US$ 98.60 in the conventional market, which we can consider to be cheap. All components for assembling the glove can be easily found in the market. This is an advantage in the proposed system, since the glove can be easily reproduced and assembled.

Table 5: List of components for the glove prototype.

<table>
<thead>
<tr>
<th>MATERIAL</th>
<th>QUANT.</th>
<th>UNIT COST (US$)</th>
<th>TOTAL COST (US$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glove</td>
<td>1</td>
<td>15.45</td>
<td>15.45</td>
</tr>
<tr>
<td>MPU 6050</td>
<td>6</td>
<td>2.75</td>
<td>16.50</td>
</tr>
<tr>
<td>FPGA 101 dev. board</td>
<td>1</td>
<td>60.47</td>
<td>60.47</td>
</tr>
<tr>
<td>HC05 Bluetooth module</td>
<td>1</td>
<td>6.18</td>
<td>6.18</td>
</tr>
<tr>
<td><strong>TOTAL COST</strong></td>
<td></td>
<td></td>
<td><strong>98.60</strong></td>
</tr>
</tbody>
</table>
For code debugging tests (both the System Verilog description of PAMPIUM and the communication interface software in MATLAB) there is the possibility of connecting the glove to a computer via cable, as shown in Fig. 26. In the case, wires are connected in an Altera DE-2 FPGA development board that replaces the FPGA 101. It allows a full debug of the system and facilitates the correction of occasional bugs.

We performed communication tests between the glove and a computer, with the support of equipment such as logic analyzer, oscilloscope and multimeter. Fig. 27 shows the glove test setup. The validation of the prototype at electrical level ensures its functionality and turns the glove ready for the use in the desired applications.

![Glove prototype as portable version.](image1)

![Glove prototype in debugging version.](image2)

**Fig. 25:** Glove prototype as portable version.

**Fig. 26:** Glove prototype in debugging version.

**Fig. 27:** Glove test bench.

3. APPLICATIONS

From the validated prototype, it is possible to implement several applications that use the gesture capture glove features, such as input devices, remote control in virtual reality, movement control for physiotherapy, remote control of robots and drones, simulation of surgeries for medical students training, communication in sign language, etc. This section describes some of the implemented applications.

3.1 Glove as a Pointing Device

A direct application of the gesture capture glove is the control of the cursor’s movement in a computer. It is the simplest application, but it demonstrates the capability of the glove interacting with the machine. In addition, it allows access to any application that has the mouse as a control device.

Cursor movement is controlled by the index finger. The movement in the direction of the x and y axes of the screen (hand in the horizontal position) allows smooth and intuitive control of the mouse cursor. The accelerometer values read from sensor number 2,
located at the index fingertip, are converted to cursor offset and transmitted to the operating system through the interface implemented in MATLAB.

In order to have efficient navigation and pixel level accuracy, we adopt a transference function as shown in Fig. 28 [12].

![Fig. 28: Transfer function relating the velocity of index finger and the velocity of screen pointer.](image)

The transfer function relates the velocity of the index finger (in arbitrary unit mickey/s) to the actual mouse speed (in pixel/s). It is divided in four regions delimited by five points: (0, 0), (0.45, 1.8), (1.25, 5.5), (3.9, 24), (40, 570). The inclination of the first line segment produces a small physical gain in the screen, allowing precision movements and the ability to reach each pixel on the screen [12]. If the index finger movement is not sufficient to move the pointer one pixel on the screen, this value is stored and added in the next count. The gain is increased for higher speeds, in order to afford a natural movement sensation to large cursor displacement.

Mouse click is implemented moving the index finger down. What differentiates the click movement and cursor movement on the y-axis is the position of the thumb. When stretched, it permits movement of the pointer. When leaning next to the index finger, it freezes the pointer movement and the click is performed if the index finger moves down. The detection of the position of the thumb is easily performed through the z-axis value of the accelerometer in the sensor number 1.

### 3.2 Glove as a Presentation Control Device

The glove can be used as an input device for controlling the slides in a presentation. The presenter needs only to move the hand naturally to navigate the slides forward or backward. It allows a freedom in the movement of the presenter and gives more dynamics for the presentation. For this, it is possible to transform the glove into a virtual keyboard and identify two movements corresponding to the “Page Up” and “Page Down” keys.

The determined gesture for the “Page Up” key corresponds to the rapid movement of the whole hand from bottom to top in the horizontal position, as shown in Fig. 29. The motion capture occurs through the values read by the accelerometers and gyroscopes located on the index finger (sensor 2) and on the dorsum of the hand (sensor 4). Thus, the detection of the “Page Up” key is set when the value of the y-axis of gyroscope 4 (y4) is less than 0.4, the z-axis of gyroscope 4 (z4) is greater than 0.5 and the y-axis of accelerometer 2 (ay2) is less than -5. Fig. 30 shows the graph of these sensors and the identification of the conditions for the “Page Up” motion detection.

![Fig. 29: Sequence indicating the movement of the hand corresponding to the “Page Up” key gesture.](image)
Fig. 30: Data read from the gyroscope 4 on the y and z-axis (y4 and z4, respectively) and accelerometer 2 on the y-axis (ay2) and the identification of the moment when the relative movement of the “Page Up” key is detected.

Fig. 31: Sequence indicating the movement of the hand corresponding to the gesture of the “Page Down” key.

Fig. 32: Data read from gyroscope 4 on y and z-axis (y4 and z4, respectively) and accelerometer 2 on the z-axis (az2) and the identification of the moment when the relative movement of the “Page Down” key is detected.

The “Page Down” movement is defined when the hand moves in the vertical position from right to left, as shown in Fig. 31. The key is captured when the value of gyroscope 4 on the y-axis (z4) is greater than 1.4, the gyroscope on the z-axis (z4) is less than -0.5 and the accelerometer 2 on the z-axis (az2) is less than -5, as shown in Fig. 32.

The implemented detection method uses only current data for the evaluation of the conditions of its occurrence. It eliminates the necessity of a memory to store past information. However, it is necessary to apply a delay when a key is detected in order to avoid getting undesired repeated key detections until the hand returns to the idle state.
3.3 Glove as a Keyboard

Another application suitable for the developed glove is the recognition of letters from gestures, in which the user makes a certain movement and the application understands it as a key pressed in the keyboard.

We adopted the alphabet from the Brazilian signal language (LIBRAS) [13], as shown in Fig. 33, as standard for the gesture recognition. LIBRAS is the official Brazilian signal language and its use is widespread along the country. With the combination of various gestures it is possible to spell complete words and phrases. The automatic recognition of the gestures with the glove and the translation to a letter in the keyboard allows the typing in a portable device (a smartphone, for example) in an easy and intuitive way for a blind or a deaf-mute person.

To collect the data, we implemented a Matlab interface. Through this interface, it is possible to verify the differences in the data read from the sensors corresponding to each letter. Fig. 34 shows the patterns of the values of gyroscopes corresponding to the letters A, B and W. It is possible to notice a clear difference between the data behavior for the 3 letters - specially between B and W, although the gestures corresponding to these letters are quite similar.

The procedure for identifying the letters of the alphabet is made with the aid of machine learning. The classification algorithms must be trained with samples of known patterns in a supervised learning process. The model resulting from the training process is used to predict the corresponding letter of unknown samples.

Fig. 33: Gestures corresponding to the letters of the alphabet in the Brazilian signal language (LIBRAS) [13].

Fig. 34: Gyroscope patterns for the gestures corresponding to the letters A, B and W in LIBRAS.
The design space is divided in 28 classes corresponding to all letters of the alphabet plus the symbols for “space” and “idle”. The input data has 18 features, composed of the measurements of six 3-axes gyroscopes previously described, captured from the glove at a given instant.

We selected six machine learning algorithms in order to evaluate which one adapts better to our application and produces the best performance in terms of accuracy and prediction time. They are the following: Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Logistic Regression (LR) and Random Forest (RF) [15].

The LR algorithm can be understood simply as finding the $\beta$ parameters for the $F(x)$ function, which is the probability of the dependent variable $x$ belonging a given class:

$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

The class with highest probability is considered the predicted class.

SVM classifies data by finding the best hyperplane delimited by support vectors which separate data points from a given class from those of the other classes. The best hyperplane for an SVM is the one that has the largest margin between two classes. The SVM can be used to separate classes both for linear and non-linear classifiers. It can use different kernels for the similarity function, such as linear, polynomial, Gaussian or sigmoid [14].

The KNN algorithm uses the distance between two points to classify the samples. It identifies a point in the middle of each class and evaluates the Euclidean distance between this and the sample to be classified with the following function:

$$F(x) = \sqrt{(p - q_x)^2}$$

where $p$ is the point to be predicted and $q_x$ is the central point of each class. The point $p$ is then assigned to the nearest class.

Decision trees are based on a set of true/false decisions that causes the algorithm to arrive at a determined class.

Random forest is a generalized form of decision trees, assigning weights to the features in order to avoid overfitting [15].

The QDA algorithm approaches the problem by assuming that the conditional probability density functions are normally distributed. Under this assumption, the Bayes optimal solution is used to predict points as being from a class if the logarithm of the likelihood ratio is bigger than some threshold $T$.

The LDA algorithm finds a linear combination of features that characterizes two or more classes of objects. The resulting combination may be used as a linear classifier. It is quite similar to QDA, with the difference that LDA makes the additional simplifying homoscedasticity assumption that the class covariances are identical and that the covariances have full rank.

The metrics for evaluating classification algorithm performance are precision, recall and $F_1$-score. A predicted class can result in four cases: true positive (TP), false positive (FP), true negative (TN) and false negative (FN). Precision is also referred to as positive predictive value (PPV) and can be defined as:

$$PPV = \frac{TP}{TP + FP}$$

Recall is true positive rate (TPR), given by:

$$TPR = \frac{TP}{TP + FN}$$

The harmonic mean of precision and recall is the balanced $F_1$-score ($F_1$):

$$F_1 = \frac{2TP}{2TP + FP + FN}$$

The precision decays if the algorithm predicts many false negatives, and the recall decays when there are many errors in a single class [16].

The prediction time demanded by the generated model is also important for evaluating the classification method. The algorithm may be unfeasible for the application if the test time is bigger than the data update. It can be estimated by measuring the average time needed for testing a vector of input instances.

We used labeled data collected by volunteers corresponding to the letters of the alphabet in LIBRAS.
We trained the classification algorithms with a dataset composed of 10 gestures for each letter from 8 different people. This dataset was divided in a training set composed of gestures from 6 random people and a test set with the gestures from the 2 remaining people. The algorithms which presented best precision were SVM and LDA, with 94% of correct predictions. Worst result was obtained by KNN, with 88%. Fig. 35 shows the results. There is a small difference between the best and the worst algorithms, but the impact of a smaller precision is relevant if considered a high number of predictions.

Table 6 shows the comparative for precision, recall and $F_1$-score for the algorithms. Considering $F_1$-score and the recall, the SVM and the LDA presented again the best values. SVM achieved 0.93 for TPR and $F_1$, and LDA obtained 0.91 for both metrics. The high $F_1$-score demonstrates that few classes have the high error rate and that the remaining present low error rate.

Fig. 36 shows the confusion matrix for the SVM Gaussian model, in which is possible to evaluate the performance of the classifier for each class.

The glove is capable to send to the computer about 100 new sensor data updates per second. So, the prediction model must be able to perform a single test in a time smaller than 10 ms. To evaluate the prediction time, we executed 10,000 runs for each generated model and measured the average time in a computer with a 2.4 GHz i7-5500u processor, using only one thread for each algorithm.

The resulting average prediction times for each model are depicted in Fig. 37. It can be seen that the model produced by the LR algorithm is the faster one, needing only 0.05 ms for a single prediction. The LDA algorithm also produces a fast prediction model, spending 0.10 ms in average. The slower model was produced by the RF algorithm, demanding 6.55 ms for a single prediction.

These results demonstrate that all heuristics can be used for real-time prediction in the target glove. The LDA and SVM, which obtained best accuracy results, produced also very fast prediction models, with a mean of 0.10 ms and 0.14 ms for a single prediction, respectively. It demonstrates that both heuristics, together with the LR algorithm, are good candidates to be implemented in dedicated embedded systems with limited hardware resources, although other aspects need to be investigated, such as the requirements of memory and arithmetic functions.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PPV</th>
<th>TPR</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN</td>
<td>0.88</td>
<td>0.87</td>
<td>0.87</td>
</tr>
<tr>
<td>RF</td>
<td>0.91</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>QDA</td>
<td>0.92</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>LR</td>
<td>0.93</td>
<td>0.91</td>
<td>0.91</td>
</tr>
<tr>
<td>SVM</td>
<td>0.94</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>LDA</td>
<td>0.94</td>
<td>0.91</td>
<td>0.91</td>
</tr>
</tbody>
</table>
With the trained model it is possible to perform predictions of new gestures. The prediction procedure flowchart is depicted Fig. 38. It receives the gyroscope data read by the 6 sensors referring to a certain static gesture and verifies if the hand is stable (no movement) prior to classifying. If the hand is not stable, the algorithm returns to reading the sensors again. If the hand is stable, the algorithm performs the classification. For the classification, the gyroscope information is the input of the trained model, which returns the class with the higher probability of containing that gesture. From this comparison, it is possible to attribute the new data to the predicted class. Finally, the predicted letter of the LIBRAS alphabet, corresponding to the given gesture, is printed on the screen.

![Fig. 36: Resulting confusion matrix in the training procedure using SVM Gaussian model. The heat map indicates the precision for each class in a scale from 0 to 20.](chart1.png)

![Fig. 37: Time necessary for a single prediction.](chart2.png)

3.4 Glove as a Joystick

The glove also has applications in the world of digital games. We can control the characters with the hand taking advantage of the cursor movements, as described in section 3.1.

We implemented a game based on the *fruit ninja* game [10] for demonstrating the usability of the developed glove. The objective of the game is to cut fruits with the character - in this case, the point of the cursor. When the fruit is hit, it appears as cut and the score is increased. The character follows the movement of the index finger. Fig. 39 shows a screenshot of the developed game. The ease of movement allowed by the glove makes the control of the character much more agile and intuitive in comparison to a conventional joystick.

![Fig. 38: Flowchart of the prediction procedure.](chart3.png)
3.5 3D Hand Model

The full glove capability can be demonstrated with a hand 3D model. This virtual model reproduces the real hand movement by means of the values read from the accelerometers and gyroscopes. Each finger movement is independent from the others, which provides a realistic control of the 3D model.

The 3D model was implemented in Unity framework. It is composed of bones for the assembling of each part of the finger. Fig. 40 shows the figure of the model of the hand, where the bones are indicated in white.

The bones are connected to each other so that the articulations reproduces a realistic hand movement. The fixed reference is located at the wrist.

This model is a prototype for any project using a 3D space, like virtual reality games, simulations or remote control. Fig. 41 shows the figure of the rendered virtual 3D hand controlled by the glove in the Unity framework.

Unity is a very popular tool for building games. It makes it easier to work because much of the game scenario is done by dragging and dropping objects. In addition, it provides native methods for the control of the hand’s bones, which allows the fingers to move a given distance while keeping the hand integrity.

The 3D hand control flowchart is depicted in Fig. 42. The movement of each finger is calculated by the difference between the current and the previous values read by the related gyroscope. This difference is added to the current position of the bone at the fingertip. Each finger has three phalanges, except for the thumb, which has two. The position of the bones are estimated in order to give the impression of hand closing or opening. This estimation is made by the multiplication of the bone position by a coefficient of the calculated gyroscope difference. The coefficients were defined as 1.3 for the distal, 1 for the middle and 0.7 for the proximal phalange.

The movement of the entire hand is calculated according to the value read from the accelerometer located at sensor 4 on the posterior part of the hand.
To catch an object, we use a collider as a trigger that is activated when the hand is open and it closes near to an object. The object local axes become child of the hand, and follow the hand movement. When the hand is opened, the object axes stop being a child, causing the hand to release the object.

3.6 Smartphone interface

We developed a dedicated smartphone application for the communication with the glove. It allows a practical use of the device, providing full mobility to the user. The application was developed in Android Studio and the communication protocol was adapted to this platform as a running class in Java language. So, the data from the sensors are directly available for any other class.

Two activities were implemented in the application, one for raw data visualization and other for the classification procedure. Fig. 43 shows the screen of the accelerometer data visualization. It is possible to choose the sensors that will be visible. Each sensor has a corresponding color, facilitating the identification in the graph.

![Graphical visualization of accelerometers data in the smartphone Android application.](image-url)
Fig. 44 shows the corresponding screen for gyroscope data visualization. In this example, it is possible to notice that the data from sensor 1 (green) is not shown when we uncheck its respective checkbox.

Fig. 44: Graphical visualization of gyroscopes data in the smartphone Android application.

Fig. 45 demonstrates the screen interface for the keyboard prediction mode for Android. The procedure is the same that was described in section 3.3, with the same trained model but with the algorithm rewritten in Java and with the parameters loaded from a text file. The user makes the gesture corresponding to a given letter of the alphabet, the prediction algorithm understands it based on the gyroscopes data and print it on the screen. When a complete word is spelled, the application proceeds a text-to-speech function, thus speaking the word.

Fig. 45: Gesture identification as a letter of the alphabet in the smartphone Android application.

3.7 High School Applications

The developed glove is a platform that allows open access to all its content, from the hardware architecture to the interface software in Matlab. This characteristic turn it suitable for a variety of interactive didactic applications. Thus, we intend to make available to high schools a teaching kit containing the hardware description codes, base codes and the guide of how to assemble and compile the project. The student does not need to have previous knowledge about computer architecture or algorithms at first moment, but he is stimulated to use the glove both to learn and to develop new applications. Some proposed activities are described below.

Programming concepts can be explored in schools from the development of computer programs that use glove sensor data. For example, it is possible to create a program to check the values of accelerometers for different hand positions and movements. The
algorithms applied to the glove can simply perform reading and visualization of the sensors data, or even evaluate the treatment of the data and its translation to useful information, as movement recognition.

The interaction with a three-dimensional space allows the student to interact with the glove by viewing the sensors data graphically on the computer screen, and discover how the accelerometers and gyroscopes behave. Physical concepts can be explored. Also, the data can be transformed into vectors, thus obtaining a notion of analytical geometry in a practical way.

In addition, interaction with real data enables testing with a three-dimensional space in real time. This enables the visualization of concepts such as object translation and rotation.

Because PAMPIUM hardware description is fully open and configurable, this allows anyone to access and modify it. So, the glove can be used as a didactic tool for teaching computer architecture. Students can explore the changes in memory size, register word length or even the size of the register bank, and check the effect on the application as a whole. The code running in the PAMPIUM instruction memory is also customizable to optimize sensor reading or to add some other functionality to the system.

4. CONCLUSION

We developed a glove for gesture capturing based on accelerometers and gyroscopes. The communication between the glove and a computer is wireless, through Bluetooth protocol, providing a good mobility to the user. It is a low cost system and can be easily used by any person with little training time. The project is fully customizable and includes a dedicated microcontroller developed by our group (PAMPIUM microcontroller), as well as the communication interfaces for I2C and RS-232 protocols. The digital control part embedded in the glove is synthesized in FPGA. The used material for assembling the glove is easily found in the market and the total cost of the prototype is about US$100.00, which can be considered cheap.

The development of applications such as mouse movement, joystick control, key recognition and 3D model shows the potential of the glove as an input device. The six degrees of freedom provided by the glove allows a natural movement detection.

The glove is specially suitable for blind or deaf-mute people to interact with a computer or smartphone in a more efficient way. The social impact is clear, since the proposed system can mitigate the difficulty of people with physical deficiency in communicating.

This project demonstrates that it is possible to implement a cheap electronic system with customized FPGA and ordinary sensors for improving the human-computer interface.

REFERENCES


