MIMICKING THE BODY BEHAVIOR WITH A ROBOTIC ARM

Borges Amanda B. , Godesiabois-Galand Thomas and Clemens Guillaume, *Member*, IEEE

Abstract - This study designs and develops all the useful components of a bionic arm robot. The core technologies are MyoWare™ Muscle sensors, Nextion display, ESP8266 microcontrollers and a robotic arm with six degrees of freedom. There is also a Wifi communication between subsystems and a man-machine interface will be designed to be able to drive a robotic arm. To do the acquisition of the human arm signals, it will be used electromyography technology and the Nextion display to interact with the robotic arm in real-time. The validity of a virtual simulation will be also studied.

I. INTRODUCTION

A. Context

The increasing use of prosthesis makes the bionics approach very compelling to replace lost limbs. This biomedical field has two big areas, the lower limbs, like legs and foot, and the upper limbs such as fingers, hand and arm. In this paper, we decide to provide the necessary components in the development of bionic arm using electromyography technology [1].

The electromyography is an electro diagnosis technique for recording and detecting the electric potential generated by skeletal muscle cells, called EMG signal, when these cells are electrically or neurologically activated.

B. Objective

The main objective of this project is to provide a robotic arm for amputees to assist them in their daily lives. Also, this project must be accessible for everybody, simple, flexible and cheapest as possible. To meet such requirements, it is proposed to use a data acquisition based on electromyography using EMG sensors, a robotic arm that mimics the movements of the real arm and a simulator to show this movements in the computer screen.

The full system must be calibrated and tested very carefully to provide the most realistic and fluid movement. Indeed, the interferences caused by external environment such as magnetic and electrical fields or other electrical signals from the body are out of the scope of this work.

The remainder of this paper is organized as follows: Section II describes the previous similar projects; Section III introduces the components and the EMG signal acquisition of the human machine interface; Section IV is about the robotic arm control and the wireless communication; Section V offers solutions for the validity of virtual arm simulation; Section VI offers conclusions and possible improvements.

System Overview

The whole project schematic shown at the figure 1 and is divided by three main areas: the signal acquisition, the robotic arm control and the simulator.

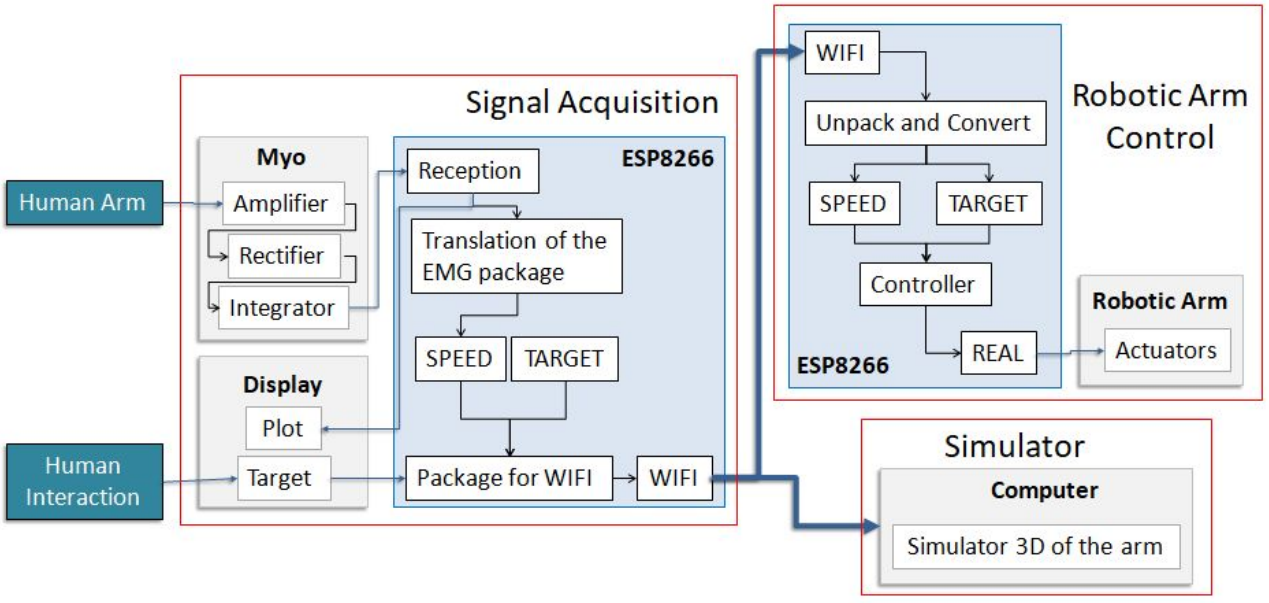

Fig. 1. Block diagram of the entire project: signal acquisition, robotic arm control and simulator. This Diagram shows three different types of data transmission: the internal ones in black; the external between devices and human interaction light blue; and the WIFI communication that is dark blue. Two different types of devices: the microcontrollers in darker blue and the actuators/sensors/simulator devices in lighter blue.

The signal acquisition is basically composed by some muscle sensors, a display and a features decoder. The muscle sensors, MyoWare™ Muscle Sensors [2], will measure, filter and rectify the EMG signal of the sensored muscle. The display, the Nextion NX4024T032[3], is a tactile human machine interface, HMI [4], to configure and control the system.

A wireless link was considered as the best alternative to come up with a standalone sensor device. For that reason, the signal acquisition and robotic arm subsystems were implemented using the NodeMCU 1.0V ESP8266 device, which provides a WIFI link between them and can be programmed using the Arduino IDE.

The robotic arm subsystem is composed by two main elements, the robotic arm and the motor controller algorithms. The robotic arm [5] used was a six degrees of freedom robot [6] which has six servomotors [7] connected by stems of metal that simulates the distribution of a human arm. They all are powered by an external provider and their ground and control pins are connected to the microcontroller.

No realistically suitable software could be engineered for the simulator. However, a good amount of visualization options were explored.

II. STATE OF ART

There are many previous projects already done at SEMI [8] using the robotic arm with a lot of different interfaces: the "*Bras robotique*" project is a very basic project whose main objective was to be able to control the arm that we use. After that, the "*ArmDroid*" project propose to control the arm with the accelerometer of a smartphone thanks to an Android application. And finally, the "*Bras Robot-RPI*" project divided in two distinct parts: a Face-Tracking system with the robot arm (the head of the arm will follow the face of a person in real time) and a musical system (the robot arm will strike on bottles to reproduce a score of music).

There is only one project that concern the electromyography technology: the "*MyoGame*" project propose to control a guitar game with muscle sensors developed to allow the detection of electrical activity of the biceps when the "scratch" of the strings is stimulated. The utility of our project is to be able to combinate previous robotic arm with myosensors.

III. EMG-BASED: MAN MACHINE TEAMING

A. *EMG signal acquisition*

The output of MyoWare™ Muscle Sensor doesn't provide a RAW EMG signal but an amplified, rectified and integrated signal therefore the EMG's envelope [5].

In order to reproduce the most realistic movements possible, the received data will be processed as follows: first, define experimentally the minimum threshold from which the skeletal muscle can be considered to be electrically stimulated. To perform this, observe the average values captured by the sensor when the muscle is in relaxation (at rest) and eventually adjust the gain of the sensor according to the muscle considered. Next, define a value illustrating the speed of the movement by looking at the successive gaps between each value read on the sensors. The figure 2 show how the signal from our sensors will be analyzed:

1) The muscle group is at rest and the threshold value isn't reached. The speed value is equal to zero corresponding to a return to the initial position with the lowest possible speed.

2) The muscle group is little excited and the threshold value is reached and the curve increases slowly. Now, we can define a speed value ranging from one to ten representative of the real speed of the motion.

3) The muscle group is solicited consistently by a load and the threshold value is still exceeded but the curve is steady. If it exceeds few seconds, the speed value will be equal to zero and a motion to initial position.

4) The muscle group is very excited and he threshold value is always exceeded and the curve increases sharply. The conclusions are the same as point two but the value of speed should be higher.

5) The muscle group is no more excited and the curve decreases, after few seconds we can conclude that the speed value is equal to zero.

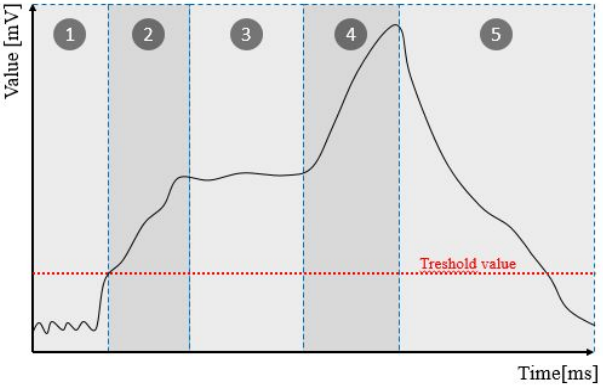

Fig. 2. The EMG envelope.

B. *Multiple EMG sources*

In order to imitate the main movements of the arm, we need to place a sensor on the biceps, the lateral shoulder and the big dorsal. However, the ESP8266 has only one analog input, this one corresponds to A0 but it is simply named "ADC" (Analog to Digital Converter). In order to control several sensors, we will use the AD7999 card which will allow us to acquire up to 4 new channels.

C. Man-machine interface

A Graphical User Interface (GUI) is used to monitor and control the robotic. The Nextion display was chosen because it's a resistive touchscreen and easy to build thanks to his dedicated editor: Nextion Editor. A welcome screen will allow us to choose between the claw and arm operations.

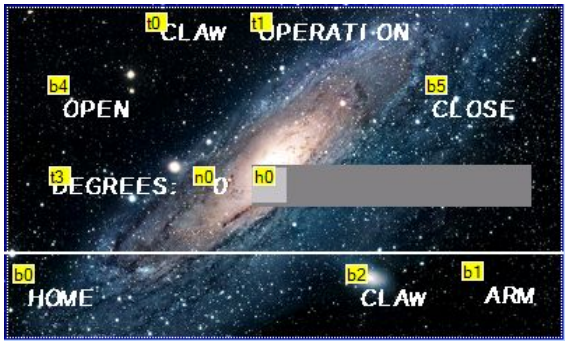

Fig. 3.1. Our Nextion Display-Claw operations.

The figure 3.1 shows us the claw operation page. Three operations is possible to control the claw: open, close or rotate (in degrees) the claw.

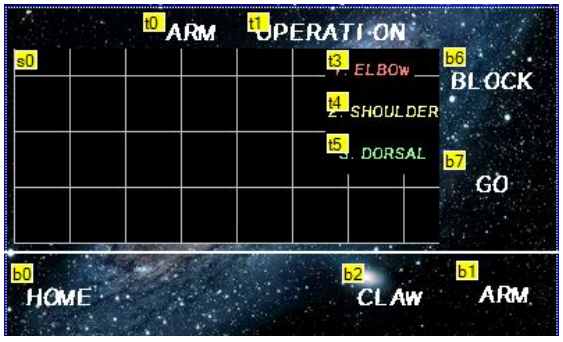

Fig. 3.1. Our Nextion Display-Arm operations.

The figure 3.2 shows us the upper arm operation page. The envelope curve of each sensor can be observed in real time on the graph. If we want to stop or resume the movement, we can press the "BLOCK" or the "GO" button.

IV. WIRELESS ROBOTIC ARM CONTROL

For the wireless robotic arm control, two work fronts were established: the control of the arm, which is how will be done the movements of the robot, and controlling it; the wireless communication that settles the communication between the arm signals and the robotic arm.

A. Control of the Robotic Arm

The robotic arm must be able to reproduce the movements of a human arm through its six servo motors. So, the control system assignment is to translate the command received from the acquisition system into a smooth movement.

Parameters

There are three main parameters, two input signals (SPEED and TARGET), that comes from the acquisition system, and one output (REAL) which will be sent to the servo motors of the arm. Every single servo motor has its own SPEED, TARGET and REAL.

- SPEED: related to the contraction of the muscle and will also be used as a velocity parameter. It is a signal ranging from 0 till 9, that 0 means the muscle is relaxed and 1 to 9 means contractions from weak to strong.
- TARGET: points to the aimed angle of each of the servo motors. It ranges changes according to each servo motor but will fluctuate between 0 and 180 in most cases.
- REAL: the angle written at each servo motor. This value will range middling from 0 till 180 and it is based on the values of TARGET and SPEED. For writing the values in the servo motors, it is used a library called Servo.h that provides the function write() that converts this angles to PWM[9] signals and send it to each associated servo.

Control Method

The control method chosen was simple and uses the parameters SPEED (muscle contraction), TARGET (aimed value), REAL (servo motor value) and INIT (initial condition of the servo motor).

As seen at the figure 4, the controlling system receives the information of SPEED for each servo motor and checks the actual value (REAL). If it is 13, the TARGET gets REAL, meaning that the servo motor will stop where it is.

If SPEED is different than zero and 13, there are two things that can happen: if REAL is smaller than TARGET, REAL is incremented with the SPEED; otherwise REAL will be equal to the sum of REAL and TARGET divided by two. For this case, both movements aims the servo motor going to the TARGET value, the first one increasing and the second one decreasing.

On the contrary, if SPEED is equal to zero, the REAL value is reduced by one until it reaches the initial value, INIT, that is when the muscles are totally relaxed.

```
for (n = 0; n < 6; n++) {
    if (SPEEDS[n] = 13)
        TARGET[n]=REAL[n];
    else if (SPEEDS[n] > 0 & SPEEDS[n] < 10)
        if (REAL[n] > TARGET[n])
            REAL[n] = ( REAL[n] + TARGET[n] )/2;
        REAL[n] = REAL[n] + SPEEDS[n];
    else
        REAL[n] = REAL[n] - 1;
        if (REAL[n] < INIT[n]) REAL[n] = INIT[n];
```

Fig. 4. Computation of the controlling signal.

B. Wireless Communication

It will be implemented a wireless communication for linking the electromyography signals of the human arm obtained at the acquisition system with the robotic arm movements.

To pass the information of SPEED and TARGET from que acquisition system to the robotic arm and simulator it was chosen the protocol IEEE 802.11 [10], called Wi-Fi, that will be provided by the microcontroller. For the transport layer protocol was selected the UDP [11] because it is fast, easily implemented, has multicast communication, low-overhead transmission and it is acceptable to lose some data samples, even though there are other protocols [12] that can be accepted as well.

For making the communication, it is necessary to settle the configuration of IP address, the port, the type of the device and the name of the Wi-Fi for all the devices, clients and server. For doing so, the ESP8266 provides libraries such as ESP8266WiFi.h and WiFiUdp.h.

Because the ESP8266 Wi-Fi communication just sends characters data type, each pair of SPEED and TARGET needs to be converted, since the acquisition system works with byte data type and the robotic arm and simulator works with integer data type.

The Wi-Fi server will be the acquisition system and the WIFI clients, the robotic arm and the simulator. That happens because both robotic arm and simulator works based on SPEED and TARGET, that the acquisition system generates. So, the acquisition system is always transmitting packages and the other two are always listening.

Data Transmission

For transmitting packages, it is important that the Wi-Fi server establishes the channel of the communication and starts it with a port and name. Once everything is settled, the acquisition system is ready to send packages with SPEED and TARGET. For doing it so, all the TARGET and SPEED values for all the motors will be organized and wrapped together in one UDP package.

The package will be an array of twelve characters that the first 6 (0 to 5) represents the SPEED and the last 6 (6 to 11) represents the TARGET to be used for every single of the servo motors in order, meaning that for every servo motor, its SPEED and TARGET value will always be at the same spot of the array as it is visualized at the figure 5.

	SPEED						TARGET					
Servo Motor	1	2	3	4	5	6	1	2	3	4	5	6

Fig. 5. Wrapping of the transmitted package.

Once the package is done, it can be sent with Udp.beginPacket() that opens a packet, Udp.write() that writes on it and Udp.endPacket() that sends it.

Data Reception

To initialize the reception of the data, it is important to pay attention on the channel of the communication that was settled at the server, that must be the same. If this gets wrong, the server and the client will be in different channels meaning that the server will send data but the client won't receive.

To receive a package, the first operation done is a checking if there is a package. It is necessary because the client is always listening. So, the Udp.parsePacket() will be used as a boolean to just call the operation of the reception of the package if it is received any, in other case, the operations of the unpacking won't be necessary.

To read the information on the package, the Udp.read() is used. After it, it is required the unpacking of the package and the conversion of the data type. Because the package that comes from the acquisition system is always the same size and has the same format, the client has a for() function that will go through the array separating the information and changing the data type to integers so the robotic arm control and the simulator can use the data.

V. VIRTUAL REALITY: ARM SIMULATION

A. *Goals*

For various reasons including ease of maintenance and customizability, it was decided to explore 3D simulation and remote monitoring solutions and try to implement them alongside our physical robotic arm.

A diverse panel of 3D software and libraries was considered and analyzed, with the emphasis being put on their viability and pertinence towards our project.

Requirements

We are looking for real-time 3D rendering software with support for two important characteristics:

- Procedural animation and control: we need to be able to alter 3D model positions and angles depending on values transmitted by the Arduino/ESP device.
- Message processing: we need to be able to communicate with the device and process the relevant information during rendering.

Moreover, as most 3D rendering software generally offers way more features than we will ever need, minimizing the amount of overhead required to have these two basic characteristics running can become a challenge.

Otherwise, every absent feature will need to be implemented and it is not obvious whether or not this is realistically possible depending on each case.

Game Rendering Software

When it comes to real-time 3D visualization and interaction, most modern game engines come fully equipped to fit our purpose.

Any sufficiently recent 3D game engine with multiplayer capabilities could potentially fit our requirements: skeletal animation, which we can potentially use to transform the model, has become an industry standard and multiplayer capabilities require message handling, generally in the form of a client/server UDP communication stream to exchange game state information and player inputs.

However, it is clear that fully fledged game engines are not suitable due to their inclusion of game management features that are, from our standpoint, utterly useless and bloated. Thus, our attention has shifted to graphical rendering engines typically aimed at video games.

OGRE [13], standing for "Object-oriented Graphics Rendering Engine", provides interesting features as a scene manager, but documentation isn't abundant and compilation difficulty is prohibitive on Linux systems. Indeed, compiling the OGRE libraries themselves required arcane tricks and we were not able to compile basic example programs, mostly because of obscure library conflicts. For this reason, standalone usage of OGRE doesn't seem to be suitable for our project at this point in time.

Blender [14] is the most well-known free and open-source 3D modelling suite available. It provides industry-grade scene management and modelling tools and can export to many file formats. General 3D rendering is performed by off-line algorithms and is thus not in real time, but the program integrates a basic game engine (BGE, for Blender Game Engine) providing real time rendering and the same Python based scripting as the rest of the suite. However, scripting documentation is relatively sparse and every attempt at getting a decent scene running have been met with texturing and lighting errors caused by what we identified as OpenGL compatibility issues. In any case, networking capabilities for BGE aren't realistically feasible or exploitable at our level.

All the technical problems we encountered lead us to think that this kind of software does not realistically meet our requirements and simply isn't worth the trouble, although examination of more game rendering software may prove interesting, especially for projects that involve gameplay mechanics. It can also be expected that future updates to OGRE or BGE might solve the encountered issues.

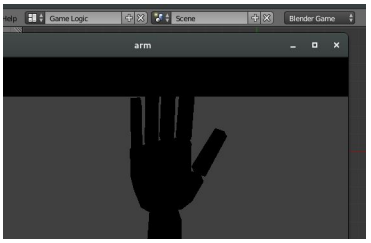

Fig. 6. Lighting and texturing errors in a test real time rendering of an arm model in Blender Game Engine.

ROS and Gazebo

Robot Operating System [15], ROS in short, is a collection of software designed to ease robotics development and provide a layer of abstraction for heterogeneous hardware clusters, i.e. interconnected devices of different nature, which includes robots.

ROS provides a master node, who is tasked with managing all other nodes and coordinate them, and slave nodes each corresponding to a specific program performing specific tasks and running on ROS. Nodes possess inputs and outputs on which they can publish or listen to messages, allowing for easy communication with any part of the ROS node network it is running on.

ROS nodes are generally created in a special development environment, namely a Catkin workspace distributed with ROS, in C++ or Python, and can be encapsulated as ROS packages for distribution.

In order to interact with ROS, a device simply needs to run the appropriate software: ROS nodes on a system running ROS or the appropriate API's for each device.

Naturally, such a feature is very interesting as one can easily create a UDP or serial message handling node, which can receive values from the Arduino/ESP devices using a ROS API then pass them on to the 3D visualization software with ROS messages.

This visualization and simulation of robotic hardware is generally performed in the context of ROS by a program called Gazebo [16].

Gazebo, for all intents and purposes, is a 3D robotics simulator. It is compatible with robot simulation formats such as URDF, meaning that in order to simulate our physical robot arm in Gazebo, we are required to create a URDF compatible specification of it: a 3D model of each segment of the arm, linked to their inertial and constraint parameters.

Interestingly, Gazebo uses OGRE for 3D rendering and has access to many different physics engines such as ODE, Bullet or Dartsim to compute scene dynamics.

The most interesting feature that Gazebo has to offer is that it can work as part of a ROS network, with its own inputs and outputs, allowing for easy interfacing with any connected device using ROS.

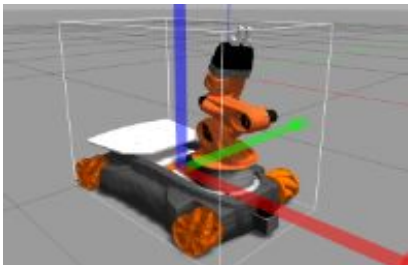

Fig. 7. URDF version of Youbot robotic manipulator included by default in Gazebo. Every part of the robot (called a "link") is interactive and is described by its 3D model, linking info, inertial and collision parameters, in order to provide a physically accurate real time simulation of the device.

All things considered, ROS seems like the perfect tool for the job. Unfortunately, using it requires integrating it in our devices and the necessary API's prove to be a considerable burden on their memory. Moreover, it has been decided to settle on a very simple message handling system with a minimal message format as explained earlier, which simply isn't compatible with ROS in its current state.

Gazebo might also not be the perfect visualization program for our robot arm, as it is a relatively resource-hungry program and is better suited to basic prototyping than monitoring real-time.

We advise further research on the capabilities of ROS and its components. It would also prove very interesting to acquire a robotic device, such as a manipulator, which already possesses a URDF format counterpart in order to greatly simplify any further work on this matter.

Scientific 3D visualization with VTK

We then decided to take a look at lighter, more scientifically inclined 3D rendering engines and investigate a longstanding actor in this field: VTK [17].

VTK is an open-source visualization toolkit aimed at scientific data-sets. It is generally used as a 3D renderer and image processing software and its object-oriented rendering pipeline is often the basis of advanced, more specialized visualization software.

VTK has Python bindings, allowing for easy testing and prototyping, but understanding the details behind every line of code needed to setup the rendering pipeline requires a fair deal of time and research.

Summary

Obviously, trying to code 3D rendering and message handling from scratch would be an error. However, trying to integrate one in specialized software for the other proves to be the source of at least as much hair pulling.

While we have a very good candidate with ROS, it seems that the current visualization options, namely Gazebo, are too cumbersome to be viable.

A good compromise would be to create a ROS node which would fetch messages from a messenger node linking the former to our devices. The rendering node could very well use VTK as is to render the simulated devices, as a sort of replacement to Gazebo. Unfortunately, this is well outside the scope of this project, but it may prove interesting to create a lightweight robotics simulator for basic monitoring, troubleshooting or customization purposes.

Finally, we would like to reiterate that acquiring a device which already has a 3D simulated equivalent would greatly speed up any related work and overall produce a better looking output.

VI. CONCLUSION

This project aimed to provide a design from all components of a bionic arm prosthesis based on electromyography. A main characteristic of this design is to connect parts dedicated to signal acquisition on the arm itself and the prosthesis via WiFi communication. As such, this allows a wider range of applications for the device than simply rehabilitating patients suffering from the loss of their limb, such as providing safer work alternatives or allowing for humans to perform actions on otherwise physically unreachable implements.

All informations and material pertainings to this paper (explanations, codes, video of demonstrations, etc) will be made available on a github page available on the web at this link [18].

Looking further, custom sensors and an actual usable physical model of the prosthesis could be designed and built for regular use. Promising developments in the field of robotics and cybernetics, such as ROS, could be integrated in the device, taking part in the great advance of man-machine interactions that the world will no doubt experience in the near future.

REFERENCES

[1] M. Roberto, P. Philip A., "Electromyography: Physiology, Engineering, and Non-Invasive Applications" *IEEE Press Engineering in Medicine and Biology Society, Sponsor.*

[2] "MyoWare™ Muscle Sensor Manual". Available: http://www.advancertechnologies.com/p/myoware.html

[3] Nextion display datasheet. Available: https://nextion.itead.cc/resources/datasheets/nx4024t032_011/

[4] "Definition of a human-machine interface". Available: https://whatis.techtarget.com/definition/human-machine-interface-HMI

[5] H. Julien. (2011, october). Construction d'un bras articulé 6DOF. *pobot.org*. Available: http://pobot.org/Construction-d-un-bras-articule.html?lang=fr

[6] Service d'Electronique et de Microélectronique of University of Mons. Web site Available: http://www.semi.fpms.ac.be/

[7] Anaheim Automation, Inc. *Servo Motor Guide*. Available: http://www.anaheimautomation.com/manuals/forms/servo-motor-guide.php#sthash.amN0Iwti.dpbs

[8] Service d'Electronique et de Microélectronique of University of Mons. Web site Available: http://www.simius.be/

[9] "How Do Servo Motors Work?", Available: https://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html

[10] L. Wolter, H. Vic & G. John, "The Innovation Journey of Wi-Fi: The Road to Global Success". *Cambridge University Press*, 2011.

[11] Gurdeep S. Hura, Mukesh Singhal, "Data and Computer Communications: Networking and Internetworking" (Book style). *CRC Press,* Boca Raton, Florida, 2001, pp. 620–621.

[12] P. Lydia, B. T. David, D. Chuck, F. Jason, L. Wei, M. Carolyn & R. Nicolas, "TCP/IP Tutorial and Technical Overview", *IBM International Technical Support Organization*, December 2006. Available: https://www.redbooks.ibm.com/redbooks/pdfs/gg243376.pdf

[13] Object-oriented Graphics Rendering Engine. Available : https://www.ogre3d.org/

[14] Blender free and open-source 3D creation suite. Available : https://www.blender.org/

[15] Robot Operating System. Available : http://www.ros.org/

[16] Gazebo robotics simulator. Available : http://www.gazebosim.org/
[17] Visualization Toolkit. Available : https://www.vtk.org
[18] Our github website. Available: https://l.facebook.com/l.php?u=https%3A%2F%2Fgithub.com%2FMandyBB%2FMIMICKING-THE-BODY-BEHAVIOR-WITH-A-ROBOTIC-ARM&h=ATMJfQ0qqsEOsJmhDkGQ0fPexvJV9GvkuwT4J-fS43NDdH5IB7Ofj2kf4G7IvAXnt9RQHzukBbF3IWKxLTsBkGWPMaqelmn65opxl1QI6k1w_Tbvyhc