UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Speech recognition with the Zowi robot

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Carlos Tecles del Romero

Tutor: Carlos Martínez Hinarejos

2018-2019

Resumen

El robot Zowi (http://zowi.bq.com/) es un robot creado para el ámbito de la robótica educativa con altas capacidades de ampliación. Zowi es capaz de detectar sonidos a través de un micrófono incorporado y se puede manejar mediante una app en smartphones y tabletas por Bluetooth. Se busca realizar un sistema que interprete el sonido captado como órdenes orales mediante reconocimiento de habla en el servidor. Como paso final, se buscará que el servidor dé órdenes a Zowi basadas en el reconocimiento obtenido.

Palabras clave: voz, reconocimiento, Zowi, habla, robot, Bluetooth, interfaz

Abstract

Zowi the robot (http://zowi.bq.com/) is a robot created for educational robotics with high expansion capabilities. Zowi is capable of detecting sounds through a built-in microphone and can be operated by a mobile and tablet app via Bluetooth. It is wanted to implement a system which interprets the sound captured as voice orders through speech recognition in the server. As a final step, the server is required to command orders to Zowi based on the obtained recognition.

Keywords : voice, recognition, Zowi, speech, robot, Bluetooth, interface

Table of contents

A mi familia,
a Carlos por su gentil ayuda
y a la OPII

1. Introduction

The rush of future technology being in our daily basis usage often excludes all the available applications it can take which are out of its usual role. Speech recognition, and the features that come along, are technologies used as a norm to satisfy the requirements of larger projects which usually point to one specific domain.

When it comes to speech recognition, those features need to be developed under the scope of the project it is aimed for. Voice recognition with an external digital device is a several factors work where the problem increases along with the complexity of the voice recognition broadness, the language transcribed where the information needs to be pulled out and the interaction between the recognition and the device.

Command by voice an external digital device focus the applyment of the technology in recognizing the language to specifically remote control the functions of it. It is necessary to support speech-to-text technology to make it understandable to the command receiving device and subsequent communicate the information it is expected to follow. The device should respond to the commands to performing properly the actions given, in order to make it functionally precise.

To accomplish this, it must be considered how the information is captured, processed, analysed and communicated to the robot, which systems are involved in each phase and how it needs to be approached according to the available technologies and the actual limitations. So far, it is required to seek for ways to improve the phases while the concept result does not operate accurately.

Zowi is a commercial robot for didactic usage aimed to children, whose features allow the basic hardware and software performance to be commanded by voice. The robot is a highly modifiable micro-controlled device which can act as the visual result of the voice recognition itself and that it is used as a platform on which the technology it is applied.

1.1. Motivations

Although generally speech-to-text technology is not new, the domain it has taken is focused on assistance, an extra feature in the way we interact with home assistance devices or mobile phones.

As Zowi is a highly modifiable device, it opens the possibility to apply the technology to an educational robot which is used by children to learn and understand technology while they play with it. Taking to the robot, instead of controlling it through a mobile application, is a big step to familiarize and bring them closer to technology and robotics. When it comes to interacting to a robot and language is involved, educational purposes can be applied to the learning functionality of the robot; for example, exercises to spell, write or pronounce properly even if is about learning a different language

1.2. Objectives

The aim of the project is to command by voice an Arduino-based robot according to its pre-programmed capabilities. To achieve and develop the mentioned functionality, the idea is:

- To explore wireless network technology to properly communicate the information command.
  To discover which network technology is used and how wireless communication works.

- To study the robot's firmware.
  To understand written built-in functionalities from the open code firmware and recognize the remote command control codification used by the mobile application.

- To process the data to generate commands.
  To stablish a certain language processing rules to process the natural language voice command.

- To understand the speech-to-text recognition iAtros engine by PRHLT.
  To use it as an engine for generating natural language voice command controls.

- To understand hardware requirements.
  To look for the most efficient way to implement the recognition according to the hardware constraints.

- To choose the appropriate technology to implement and develop the integration.

1.3. Structure

The way the project is structured and presented is justified on the work methodology followed and the piecemeal approach to the problem, a staged work which is described by the points of the previous paragraph if ordered chronologically.

The following chapters of the report define the background theorical analysis on the problem including: the chosen architecture among the rest of the possibilities, the Bluetooth communication with the device, the internal command language, some text processing rules, and an automatic voice recognition insight. The front-line implementation of the project must take into consideration: how to make serial communication via Bluetooth work and how to integrate the automatic voice recognition considering the in-between data flow. An empirical optimization and testing of the recognition would be convenient as well. The final part of the report includes the conclusion with a general review, the extension possibilities of the work and a biography to append the references.

2. Theoretical background

In this chapter the theoretical questions will be discussed and solved to define the conceptual principles of the work.

2.1. Solution scheme

As a matter of clearance, the idea proposed is defined upon a practical application of a working speech recognition schema to command Zowi. To solve this problem, we need a proper audio capture and a capable processing device.

The Zowi board[1] is equipped with an analog microphone which main purpose is to sensor raw sound to respond with predefined actions. Two non-volatiles memories are used for basic device information and software storage respectively. A CPU chipset is placed on the board, inside the plastic head structure along with the microchip.

Since Zowi lacks a digital flow current of audio and the proper fast-access memory units, audio capture in Zowi is directly dismissed. Assuming that the CPU could manage all the memory requirements of the speech recognition, portability- including speech recognition libraries- is needed to be run on the device. To that end, data capture and recognition within Zowi's hardware is certainly impractical.

Processing needs to be externalized and the information regarding command orders communicated to the robot. Although a speech recognition system could be implemented in most modern mobile devices, nonetheless it leads to rewrite the voice recognition code, and to develop a software similar to the Zowi mobile application in order to command the robot via Bluetooth.

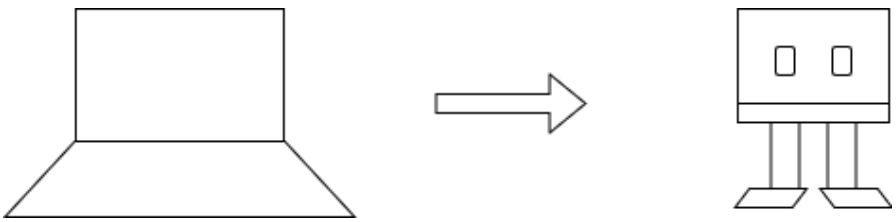

Figure. 01. Solution architecture

All things considered, processing is computed at a personal computer as the main external processing unit, as shown in Fig. 01, which will capture the audio by using the laptop's built-in microphone, recognize it, process the data and send it to Zowi via Bluetooth as a generic simple architecture.

2.2. Bluetooth communication

Current modern devices boards usually include network modules for wireless communications. In our case, Zowi has a functional capacity to be remote controlled by a company's mobile software application via Bluetooth connection. Our robot is built upon a micro Arduino chipset, a smaller single-board microcontroller version of Arduino which makes the communication possible through the HC-05 Bluetooth electronics.

The HC-05 Bluetooth module is a hardware component for transparent serial connection set up on the electronic boards as found in the documentation in appendix 1. Working as a Serial Port Profile, the chip allows Serial Port Protocol communication using RFCOMM between two peer devices. Thus, the module defines the protocols and procedures that shall be used to emulate serial cable emulation communication such as RS232[1].

As far as peripheral devices are concerned, the serial communication works as a network communication procedure where data is sent from one device to another one bit at a time, and this communication works at a specific baud rate at which information is transferred in a communication channel.

For that instance, the module is a Universal Asynchronous Receiver/Transmitter interface (UART) whose main purpose is to receive data. The interface works as an actual intermediary between our main processing device and Zowi, in which the communication is meant to carry the command data[2].

2.3. Zowi's commands

The open-code firmware[2] of Zowi is public and accessible to anyone, as well as the support precompiled libraries[3]. The arduino language is a dialect translated by the Arduino IDE and compiled as C/C++ language[3]. The language simplifies the coding and access to the information of the pins of the board, such as voltage or data stream.

The software main logic is written in arduino language and defines the basic loop logic and the robot states and functions, including the *SerialCommand* library which works as an object in the main loop. The information given from the Bluetooth module pin is read thanks to the supported libraries and within the arduino logic according to the flow control defined.

Hence, the firmware defines 4 modes of flow control, one of them is the teleoperation mode as shown in the firmware comments, where the Bluetooth-HC module pin information is read, and the data stream managed. The firmware is written in arduino and can be found in the Zowi's GitHub repositories referenced in appendix 2.

2.3.1. Serial Commands

SerialCommand[4] library manages the Serial communication at the logic level. Serial communication within library parameters treats the information streamed from the pin using a data buffer where the commands are stuck. The buffer works as a size-limited FIFO queue

and the commands are the input; then are read as characters, not as bytes. Commands in the buffer are low-level parsed by their prefix and handled along with the rest of the command string, so its related pair function are launched. The library uses the carriage return control character to signal the end of each command input in the buffer.

The available serial commands are handled at the logic level as a string command list, where commands can be added if new related pair functions are defined. *SerialCommand* communication is blind to the developer in order to make the callbacks easy to be added to the communication, making Zowi highly expansible.

Within the arduino software, although the idea is to execute the command as soon as it is processed, an ACK is sent at the beginning and at the end of each callback to feedback the status of the device.

2.3.2 Zowi command codification

To teleoperate Zowi, we need to know which command information needs to be sent to activate the proper action in the robot. That is to say, the command language it is used to choose and run the callbacks.

Among the available Serial Commands to send information to Zowi, including changing the device's name and requesting information from the sensors, three are chosen regarding the physical movements the device can perform. These commands are shown in Fig. 02.

```
Stop command       -code: S
Gesture command    -code: H GestureID
Movement command   -code: M MoveID T MoveSize
```

Figure. 02. Used serial commands

The command primary prefix defines the structure of the command codification, the following values define the rest of parameters. For example, for the movement command we can find the codification shown in Fig. 03.

```
M MoveID T MoveSize
```

Figure. 03. Movement command

In that case `MoveID` is the identification number of the movement function, the `T` is the time the movement is done and `MoveSize` is the frequency of the movement. Each callback function has its own properties; in this case each movement ID is related to a constant information on how and where the servomotors should turn.

If `T` and `MoveSize` are not given, it runs the predefined configuration passing the standard defined parameters to the method. For the rest of the commands, such as LED light screen, extra not applicable parameters are dismissed by the handlers.

Following the software comments, a brief documentation can be extracted about the command codification used to interpret the orders and run the callbacks. For more information check the *SerialCommand* commands available in the teleoperation mode commented in code, referenced in the appendix 2.

2.4. Text Processing

The language model of the speech recognition defines the list of sentences that can possibly be the output of the recognition[4]. Those are the finite set of sentences in natural language which contain information about what Zowi should do and which orders need to be communicated to the robot. To translate the transcription, the sentences need to be processed to associate natural language orders to the command codification used by Zowi.

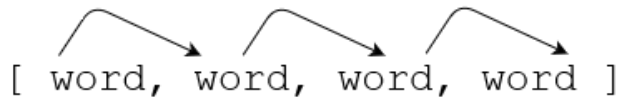

Figure. 04. Text processing

As the idea is to aim the speech recognition to children, the set of phrases is not large. The grammar language can generate different sentences that mean the same, so the general system is able to accept synonyms or words that semantically carry the same information. To that extend, the process needs to follow some logics to build the transcription and properly associate different lexemes to the related codification.

2.4.1. Pseudo-parsing

The parsing works as an iterative process where each word of the sentence, one by one, is analysed from left to write to form the associate codification as shown in Fig. 04. Words which carry information of the command are treated as significative lexemes, and two basic lexeme types are defined: "*modular*" or "not *modular*". That means, the code associated must be defined on the next *significative* word or is directly associated to an action ID. In this case, there is not a possible transcribed text generated by the recognition that contains a modular verb and it does not include a *significative* lexeme which defines which action ID to associate.

For example, the action to turn is defined by different Movement `ID` –as they are different actions - one `ID` to turn left and one `ID` to turn right, as it happens with other 5 chosen actions. Thus, "*gira*" and "*derecha*" are different lexemes, and in the translation, there is no

one-to-one relationship. Nevertheless, the recognition grammar should define where the robot turns as the robot does not randomly chose where to. The syntax therefore is simple; the modulator can be semantically understood in natural language as the adverb of the action.

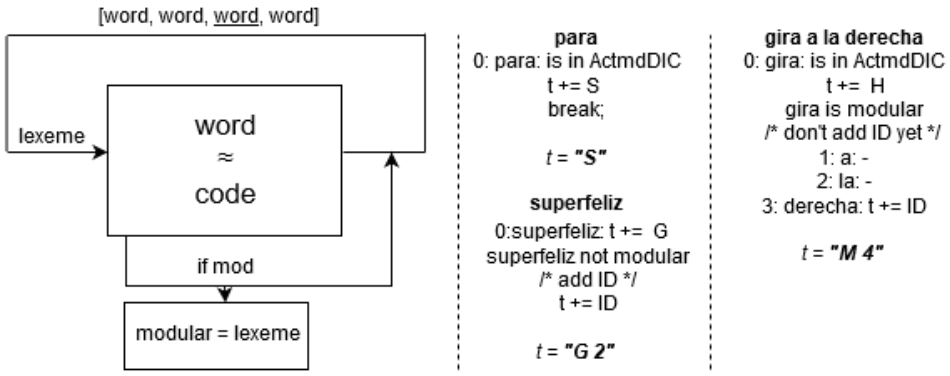

Figure. 05. Iterative process

Each word through the iterative process is checked if belongs to the lexicon of the translation. If it does not, is not considered a *significate* lexeme, so is dismissed. On the other hand, if it does, the associated code is added according to the sequential logics or the state is modified to choose the next associated code depending on if it is not modular or if it is modular respectively as discussed in the traces of the Fig. 05.

2.4.2. Dictionaries

To handle all the information, the lexicon needs to be stored relationally to access the related prefix or parameter values of the Zowi codification. This information could be accessed depending on the semantic of the word, where this is shaped by the language that models the grammar in the recognition.

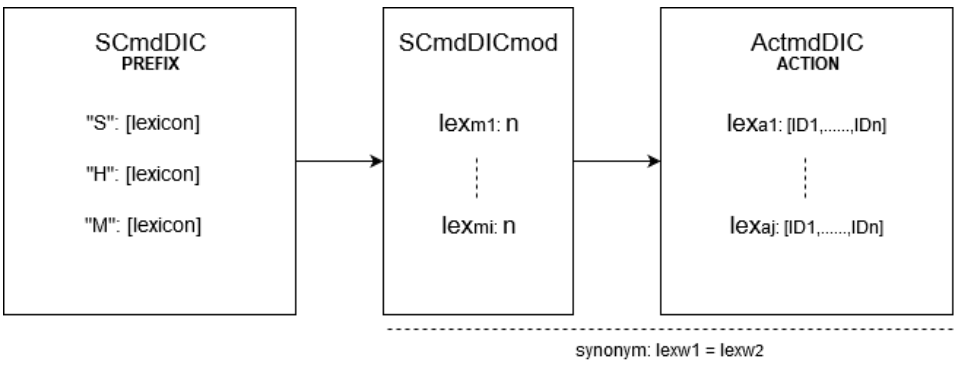

Figure. 06. Dictionaries

The items of the prefix and the action dictionaries in Fig. 06. are multi-value lists where each key is the associated code, and the list includes all the possible significative lexemes that are translated. Two equal lexemes cannot be in different lists of *SCmdDIC* so that the set of lexemes of each key is excluding. It is acknowledged the lexicon stored as an array per key can be inefficient; however, as the command set is small, to structure the data in dictionaries is clearer at seeing the relationships between the lexicon and the codification and later can be properly structured in case this information is required for future work. Significantly, as shown in Fig. 07., the *SCmdDICmod* dictionary stores the position of the `ID` in the array of values located in the *ActmodDIC* dictionary for each word which is modular.

For example, for "apóyate a la izquierda", the two *significative* lexemes are "apóyate" and "izquierda". "Apóyate" is located between the lexemes of the `M` lexicon so the translation generates a `"M"`. "Izquierda" is located between the keys of the *SCmdDICmod* so the translation generates the value located in the position of the array of the word in *ActmdDIC* pointed by the value of the *SCmdDICmod* which is `"16"`. The translation is added to the string of the codification having `"M 16"` as shown in Figure. 07. To proceed this way is based on the idea of the translation code being sequentially generated when the loop is iterating over the words rather than analysing previous and following words to apply some logics. The relationship between the dictionaries build a structure of possible combinations that can be translated from natural language to the associated codification, defining a "Zowi-like grammar", exemplified in the syntactic tree shown in Fig. 07.

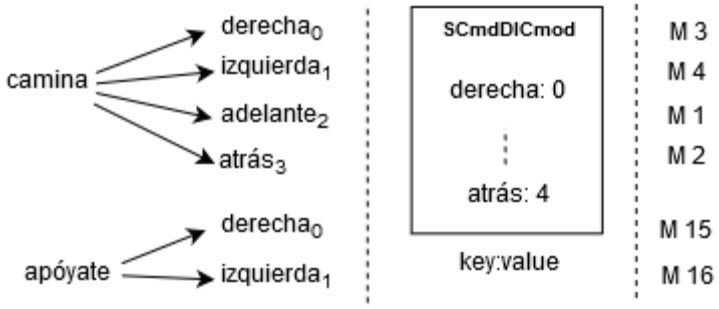

Figure. 07. Possible combinations of words in natural language to point direction

As the language and the transcription work, the relationship between lexicon and terms is stablished in the way that if the language is slightly modified to be little bit more flexible to the grammar that models the data sequence, the process and the significative lexicon stills translates the text to the proper code. On the other side, if new functionalities are applied to the holistic recognition system- meaning the Automatic Speech Recognition (A.S.R) language model is modified to generate new or extra information related to a new voice command- dictionaries can be refreshed. The lexicon accepts the word and the translation builds the related code to properly activate the new functionality, if and only if the modification in the A.S.R. grammar follows the same simple imperative grammar and information regarding the parameters of the prefix code are giving in the same order of the codification of the command.

2.5. Automatic Speech Recognition (A.S.R.)

The Automatic Speech Recognition[5] converts the audio signal of the speaker's voice into text: the recognition is understood as the transcription of the spoken language. The text generated is the text which is afterwards processed to make the recognition effective, giving it a meaning within the general recognition system, as shown in the section 2.4.1. Pseudo-parsing.

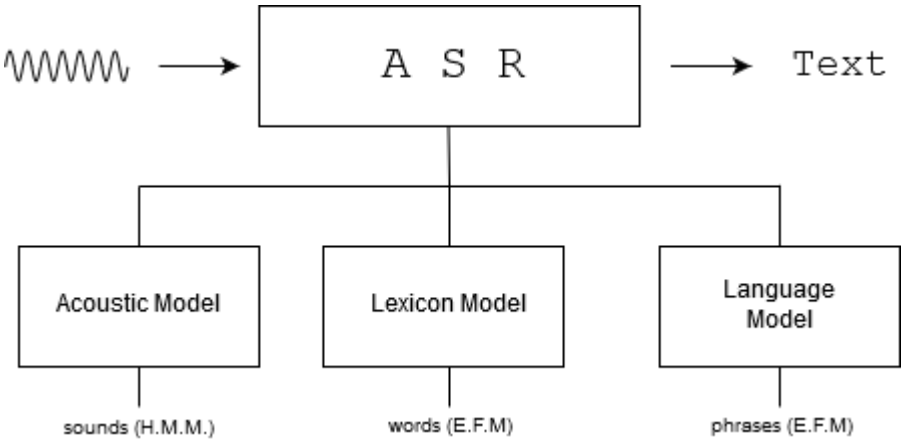

Figure. 08. ASR schema

The speech recognition is usually independent of the speaker, treating the audio data equally for any speaker's voice input in the system. Speech recognition hence is comprehended not as the identification of the speaker's voice, but as the transcription of the voice input into text[6]. In this case, it is not necessary nor useful to the Automatic Speech Recognition system to process the voice to be profiled, as the use is general and free to any speaker whose pronunciation is collected in the acoustic model.

As shown in the Fig. 08. the system is composed by three models: the acoustic model which is Hidden Markov Model-based (H.M.M.-based), and subsequent models which will shape the probabilities of the data sequence. The words and the sentences are estimations of the Markov Process understood as Estimated Frequency Models (E.F.M.).

The aim of our system is, in general terms, to generate the proper data sequence that more likely corresponds to the voice command voiced to the audio capturing device.

2.5.1. Phonetic-based HMMs

Traditionally, speech recognition models are based on phonetics. The structure it is relied in a statistical representation of the distinct sounds that make up each word in the language model. That is, the data model abstraction of the recognition defines the units of sound that shape a sentence in the grammar by the words that possibly are shaped by the units of sound, an intuitive abstraction of the model disposition shown in Fig. 09.

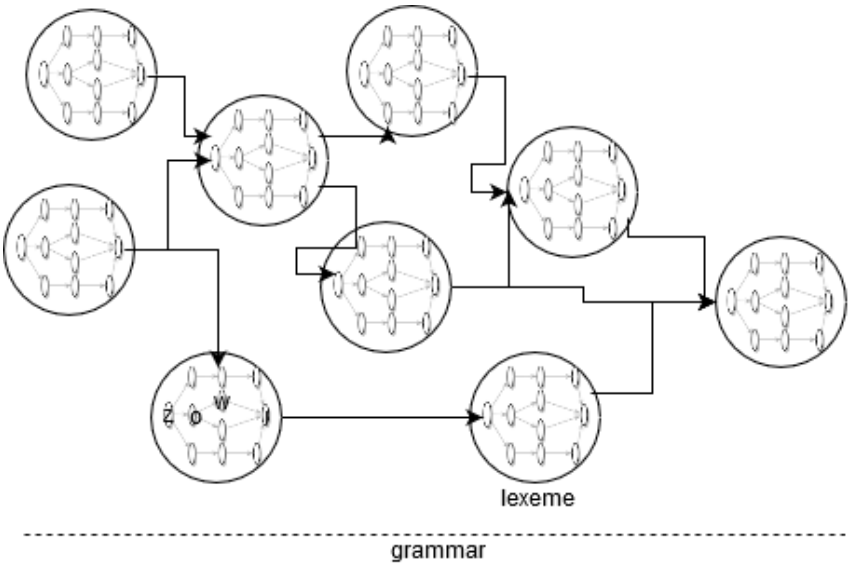

Figure. 09. Intuitive and approximate abstraction of model disposition

HMM-base systems or speech recognition systems based on Hidden Markov Models are statistical models in which the data sequence being modelled is assumed to be a Markov process with unknown parameters, where these need to be determined from the observable data[7]. The output is constructed from concatenating the individual trained models for each element of the recognition scheme, where each phoneme has a different output distribution.

$$(1)\quad \hat{W} = \underset{W \in L}{\arg\max} P(O \mid W) P(W)$$

Figure. 10. Estimated weight function

As shown in Fig. 10. the acoustic model is the conditional probability of the sequence of data given the probability of the word being shaped[8]. The estimation models will shape the different solution paths the transcription can take as a solution graph.

To decode the most likely output for the given data sequence, it is needed to maximize the output probability of each path in the set of HMMs. For that purpose, Viterbi algorithm is used to find the best solution path of the sequence[9].

In conclusion, the A.S.R. defines explicitly two models to model the process: the acoustic model and the grammatic model.

2.5.2. Lexicon

The lexicon is the inventory of lexemes that the language can accept. To build the data sequence of phonemes as words, a relationship between phonemes and words needs to be defined within the recognition system. The lexicon is the pronunciation dictionary that defines how each word of the language is pronounced as shown in Fig. 11. In this case, the phonology used is that of Spanish, as the language of the voice input is Spanish. At the end, phonemes represent the units of sound that shape a lexeme, so any lexeme could be recognized within the system if it is pronounced using the phonology defined in the recognition.

```
izquierda  1.0  i z k i e r d a
```

Figure. 11. Lexeme phonology

2.5.3. Grammar Model

Following Zowi's imperative codification as defined at the end of section 2.4.2., a simple syntax needs to be defined to generate the text and build the synonym expression. For the commanding purposes, the language grammar is regular and is defined by a finite automaton, a finite state machine that defines the finite set of phrases that possible can be modelled as a data sequence in the recognition. In our case, the language grammar shown in Fig. 12. is based on the selected orders chosen in section 2.3.2.

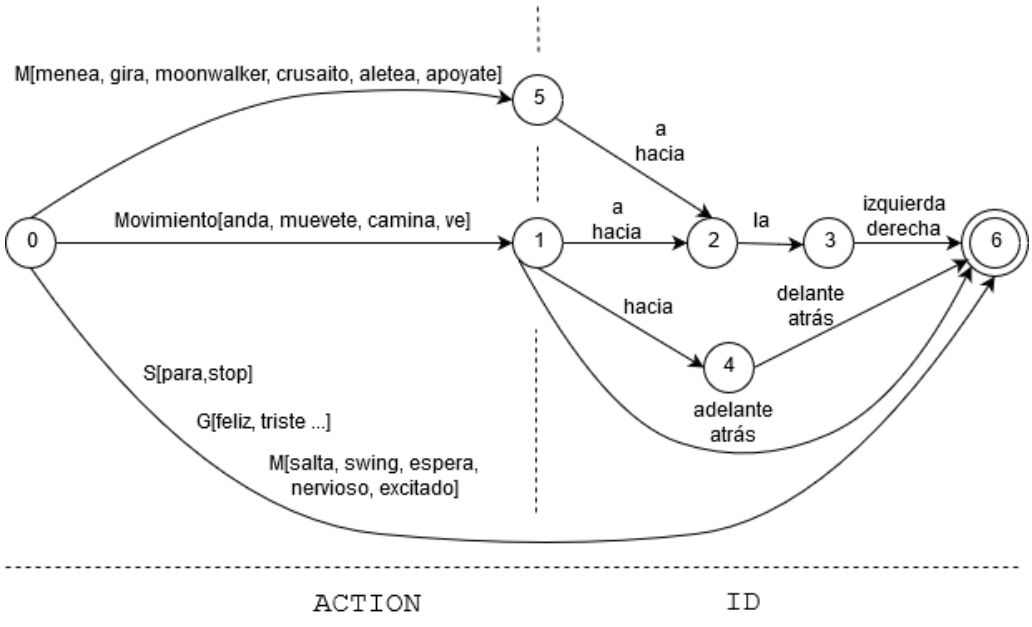

Figure. 12. Language model and its related syntax

2.6. General recognition system: data flow

To conclude, the data flow can be understood as a staged process from the voice input to the command communication.

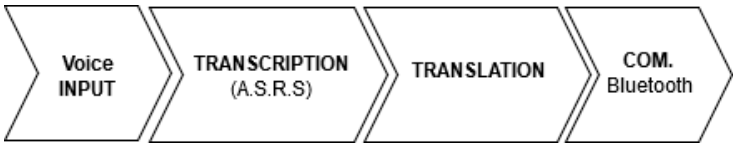

Figure. 13. General recognition system: data flow

The audio signal captured by the microphone is transcribed by the Automatic Speech Recognition System, the system converts the data from the audio file into text; the text is properly processed according to translation defined in section 2.4.1., and an associated arduino workable code from the figure shown in Fig. 02. generated. Ultimately the code is sent through Serial Communication to the device in teleoperation mode.

3. Implementation

To carry out the implementation of the project, scripting will be used for the communication and processing, along with a software of the PRHLT group[5] for the speech recognition.

3.1. Communication: with Zowi

To apply the computer to device communication, first we need to specify the concepts of the communication and how they are going to be applied.

As previously spoiled in the solution architecture that defines the devices involved in the network architecture, in Fig. 01., the laptop as the main processing unit carries the general Automatic Speech Recognition and directly communicates to Zowi the *SerialCommand* commands using Serial Communication. This communication is completely directional, as feedback is not needed. That means that, although is possible to have a bidirectional communication between both devices, the communication does not operate to respond to any feedback coming from the robot.

The serial communication is taken over by the network adapter, common to both operative systems: Windows and Linux, communication through Serial Ports is supported. Unix system is used, as the A.S.R. is based on Debian. In this case, Ubuntu distro runs on the laptop to operate the communication through the in-built network adapter.

The access to the serial ports is given by the internal network interface of the operative system. The interface is the logic layer to access the electronics of the network board and to manage the communication allowed by the Bluetooth characteristics of the adapter. To avoid messing too much, and as a A.S.R will need to be integrated within the system, a Python library is used.

3.1.1. RFCOMM communication

The network technology is wireless and radio is used by both devices as the physical network layer over which the commutation is hold. The laptop and Zowi need to be at a workable distance before stablishing the communication channel. From the processing side, among the available serial ports, one is given by the network adapter, allowing the system to access the communication interface. On the other hand, Zowi uses a modified version of the *SerialCommand* library to stablish the communication: *ZowiSerialCommand.h*. The library functionality is mentioned in the conceptual principles of the Serial Communication with Zowi in section 2.3.1.

When the arduino loop is initialized, the first statement activates the communication by using the following line: *Serial.begin(115200).* The parameter specifies the baud rate of the communication, that is, the speed of the communication over the data channel[10].

The communication is run by the device when the robot is switched on, in order to stablish the communication channel, the devices need to be paired from the computer unit. The ports used are `/dev/rfcomm0` in Unix-based systems or `/COM0` in Windows.

Once the computer and Zowi are paired, arduino responds with a status check with name of the device storage in the EEPROM, firmware version and battery level. This information thus, is read and printed in the terminal, but not handled within the script flow control.

3.1.2. pySerial

In order to access to the ports of the interface and to read the communication, the multiplatform interpreted programming language Python 2.7 is used. Python is a commonly used functional object-oriented scripting language with a vast support for modules. Modules make Python powerful by avoiding programmers to depend on a specific understanding of operative processes.

pySerial[6] is a Python module for Serial communication allowing the developer to access the operative system network interface and its functionalities. The library is open and has the community support.

Main object serial class to construct serial communication is defined as follows:

```
with serial.Serial(port="/dev/rfcomm0", baudrate=9600,
        bytesize=EIGHTBITS,
        parity=PARITY_NONE,
        stopbits=STOPSBITS_ONE,
        timeout=0,
        xonxoff=False, rtscts=False,
        write_timeout=None, dsrdtr=False,
        inter_byte_timeout=None, exclusive=None) as ser:
```

As there is no need to control the dataflow from the hardware electronics, the parameters *RTS/CTS* and *DTR/CSR* are set to False. The hardware flow control mechanism allows the receiver and transmitter to share their states in some point of the communication. *XON/XOFF* flags to request or alert about the sending or receiving state[6] are set to False as well.

If communication principles would change to make the computer and the device to exchange information, serial flow control parameters could be modified to regulate the communication from the hardware electronics of the Bluetooth module seen in section 2.2.

The constant parameters are defined as follows:

```
STOPSBITS_ONE = serial.STOPBITS_ONE
EIGHTBITS = serial.EIGHTBITS
PARITY_NONE = serial.PARITY_NONE
```

STOPBITS_ONE set the number of bits of the hardware signal to one, so the communication is feedbacked bit per bit; *EIGHTBITS* is the byte size and *PARITY_NONE* defines the absence of a parity control per each character sent.

The basic command to read the input from the serial port is as follows:

```
s = ser.read(45)          # read up to 45 bytes (timeout)
```

The statement reads the specified bytes from the serial port, as there is no timeout, it is block until at least the specified bytes are read. After the serial object *ser* is read, it prints the incoming status check from Zowi when the communication is first stablished.

The basic command to write the output to the serial port is as follows:

```
ser.write(input) # where input + "\r\n"
```

As the orders are sequentially sent to Zowi and voice capturing, and processing are slower than serial communication, it is no needed to flush the write buffer before sending a new command unless the new commands cancels a previous one. In our scenario, the stop command described in Fig. 02. breaks the functions that Zowi performs; so that, this functionality is managed in the arduino code as stop function which is a *SerialCommand*. The input parameter is a typed variable string which carry the carriage return ascii character.

3.2. Automatic Speech Recognition System (A.S.R.S.) integration

The idea is to merge the internal A.S.R process with the voice capture, the text processing and the communication. The result is a human-robot interface which process the recognition through the command line of Linux.

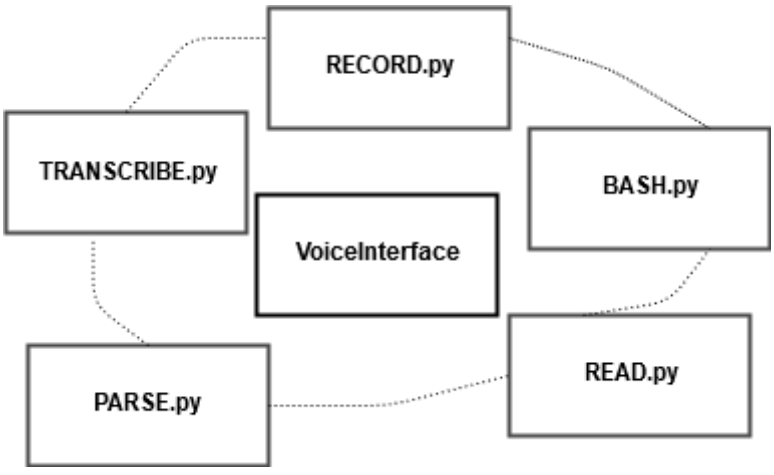

Figure. 14. General recognition system: data flow

The interface integrates the modules shown in Fig. 14. that sequentially read and write the output and input of the data, run the scripts to integrate to pre-process the audio and integrates all the phases of the recognition allowing command Zowi by voice instantly.

3.2.1 iAtros by PRHLT

iATROS (improved ATROS)[7] is a speech and handwritten text recognizer whose core is a Viterbi-like search on a Hidden Markov Model network. The software is composed of pre-processing, feature extraction and a core recognition module written in C. The version for integration purposes used is the iatros-offline version, and it is run on the laptop as processing takes place in the mentioned device. The software folder of iAtros carries two configuration files, one defines the pre-processing parameters of the audio streamed into the system and the other defines the path of the `/models` directory along with the input and output files.

In order to accomplish the integration, the software version of iAtros takes the input audio file- the cepstral coefficients file- if it exists; otherwise, it waits in a loop until it appears. After the transcription is done, removes the audio file with the cepstral coefficient information and writes out the transcribed text to an output text file. The writing-deleting of the input and the output are used as the semaphores of the integration of the iAtros process as seen in section 3.3.

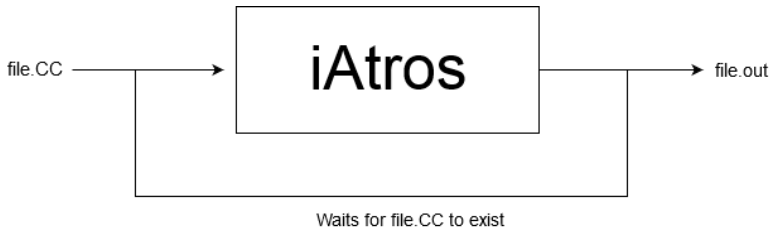

Figure. 15. iAtros control flow

The software is launched from a batch script which defines two initial statements: one to end all existing processes of iAtros and other to remove the output file. The output text file needs to be removed so the foreign processes of the A.S.R.S. do not mess with the timing and the general data flow processes properly deliver the integration.

In summary , the software launched works as a loop, where recognition waits for the file.CC to exists, carries out the transcription, deletes the file.CC, and creates the output file with the output text to be awaiting again, as shown in Figure. 15.

The models of the recognition are defined in: `./models/zowi.lx` and `./models/zowi.gr` text files; both text files contain the information regarding the lexicon and the grammar. As seen in section 2.5.2. and 2.5.3. iAtros processes the language to build the lexicon model from the information of the text on one hand, and to build the finite state machine that defines the information of the language model.

The .lx defines the words and their phonetic transcription according to the phonetic HMM model particulary. To help defining the phonetic relationship between words and their phonemes, per script *eutranscribe* is used. The .gr defines the states and transitions along with their correspondent probabilities to interpret the regular grammar.

3.2.2 voiceInterface.py

The voiceInterface.py[8] script works as the main script to launch the integration: the script carries out the main call of iAtros and, within the loop, the calls to capture, process, translate and send the communication.

voiceInterface.py runs the iAtros process in the background from the first stage after the connection with the robot is stablished, afterwards it gets into a state loop that sequentially runs the callbacks to deliver the functions of the interface.

The interface work as a loop that, like iAtros, waits until a condition is valued to true to proceed. This condition works as a semaphore to time the iAtros process with the main python interface process. Following the data flow, audio capture is the first stage of the general recognition. The capturing is carried out by sox, a cross-platform software of audio editing that is under GNU license. As the audio has a command-line interface; it works as our first visual interface in the loop. The audio is taken in mono (-c 1), one channel of audio; with a sample rate of 16000Hz (-r 16000) and from the predeterminate input microphone of the operative system (-d).

Basic command for capturing audio is as follows:

```
sox -d -c 1 -r 16000 kk.wav
```

The audio needs to be processed to convert the .wav file into the feature vector file, cepstral coefficient (CC) file, valid to be read as the input of the A.S.R.S. To convert the audio file, two bash scripts are launched in the respective order: ./wav2raw cleans the first 45 bytes and ./raw2CC converts the audio to CC data executing the *iatros-speech-cepstral* module.

The audio is recognized by the iAtros process which runs in the background from the beginning of the execution. The removal by the iAtros process of the output file, allows the voiceInterface.py process to control the flow of the execution in order to properly read the output of the recognition. The python process waits until the output file exists, as each time the recognition is done, the output is written in a new output file with the line transcribe in it.

The python process reads the output text file, which is formatted according to the iAtros configuration file in ./zowi.cnf.

Basic output text format is as follows:

```
"<s> word <sil> word </s>"
```

The brackets <s> </s> determined the beginning and the end of the transcription, and the <sil> label, the silence audio between words if it exists. The text is read and logged out in a loggin file in /readingFile/log.txt (if it doesn't exists its created automatically). This

file is parsed or filtered as discussed in the chapter and given as a list of strings. The translation module is called afterwards to give the related transcription. This data is written in the open serial port back within the main loop as shown in Fig. 16.

Any callback module exception is raised to be show in the main loop if cached in the try-catch statement with the format:

```
Error type – raised cached (E)
```

Modules callbacks are briefly described section 3.2.2., an API documentation of the functions calls of the methods.

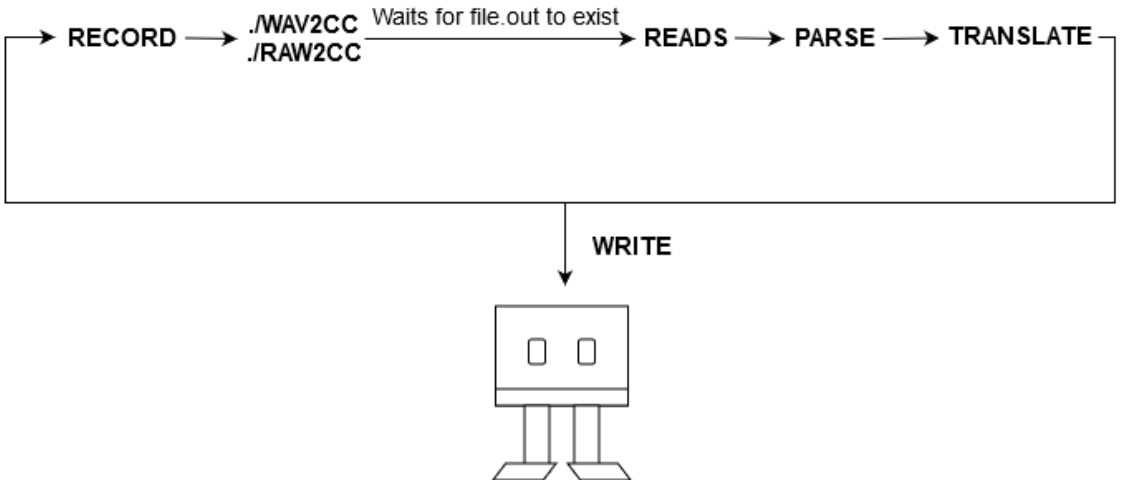

Figure. 16. voiceInterface.py control flow

3.2.3 Modules

The interface is built as an add-on of python scripts that deliver the functionalities of the voiceInterface.py script. The modules are called and divided according to the task carried to clarify the structure and give it a sense within the conceptual A.S.R.S. data flow.

The modules represent clearly the data flow and the execution of the statements. The processes that are called subprocess python library are launched as children processes and should be blocking the execution of voiceInterface.py until they are finished.

RECORD	*Parameters*	*Returns*
run(command)	`sox -d -c 1 -r 16000 kk.wav`	-
# Given a string, launch the command line in terminal and PIPE the result from the system process. Each line is read and printed by the python process and the input from the python process is sent to the launched process. If interrupted by keyboard, passes an exception.		

Table. 01. Record methods

BASH	*Parameters*	*Returns*
run(command)	`./wav2raw` `./raw2CC`	success
		failure
# Given a string, launch the command with the subprocess module. If the call results 0, it returns 'success' otherwise it returns 'failure'.		

Table. 02. Bash methods

READ	*Parameters*	*Returns*
main(path)	`/tmp/cops.out`	"<s> word <sil> word </s>"
# Given a string, reads the first line of the file located in the path and cleans the "\n" ascii character. Calls logging() and then borrar() methods.		
logging(pathlog, line)	`readingFile/log.txt` "<s> word <sil> word </s>"	-
# Given two strings, writes the line string formatted with date at the beginning of the file located in the path. Works as a logging file to keep track on last inputs of the system.		
borrar(path)	`/tmp/cops.out`	-
# Given a string, deletes the file located in the path.		

Table. 03. Read methods

PARSE	*Parameters*	*Returns*
main(orden)	"<s> word <sil> word </s>"	[word, word]
# Given a string with the ASRS output format, it returns the list of words contained.		

Table. 04. Parse methods

TRANSLATE	*Parameters*	*Returns*
main(commandL)	[word, word]	M ID
# Given a string, returns the build translation associated in the dictionaries.		
contain(element, DIC):	"word", SCmdDIC	['stop','para']
Given a two-dimensional value listed dictionary and an element, returns the list of values containing the element, otherwise returns None.		

Table. 05. Translate methods

3.3. General integration: data flow

The integration of the python script process and the iAtros process that runs the A.S.R. software is hold upon two principles to time and deploy the processes:

First, the calls to run processes from the python process; voiceInterface.py runs a child process that is executed within the interface until is completed or finished. That means when they are being executed, next line of the interface is not interpreted. Three process are launched with the *subprocess*[9] module, the process opened to run the voice recording and the call to run the two bash files (`./wav2raw` and `./raw2CC`) that pre-process the audio

file. The process *popen* allows *stdin* and *stdout* piping to share information between the main python process and the child process running the voice capture. In that case, when voice capture is finished, but not completed as it is unlimited, the user press *Ctr+C* to cancel the process and continue the iteration.

Second, the iAtros process runs in the background called by the python *os* module with the *system* statement and the ampersand option.

iAtros checks file.CC of feature vectors to apply the recognition, transcribes the output, removes the audio file.CC and waits again. On the other said the python process reads from the output and deletes it after the file is written in the log file and returned as shown in Fig. 17. There is no possibility of a new audio file, regarding a different command, saved into the system before the file.CC is deleted or the file.out removed, if the A.S.R.S. directories system is untouched.

The calls referred in the tables of the section 3.2.3. show the rest of calls that are made within the python process to read the text in Table. 0.3, to filter the transcription in Table. 0.4. and translate to the associate codification in Table. 05. The communication is sequentially made as pointed at the end of section 3.1.2.

The errors raised from each call to the child processes or from the methods used in the modules are printed in the terminal but do not break the loop; to permit to directly verbose and test the data flow of the A.S.R.S.

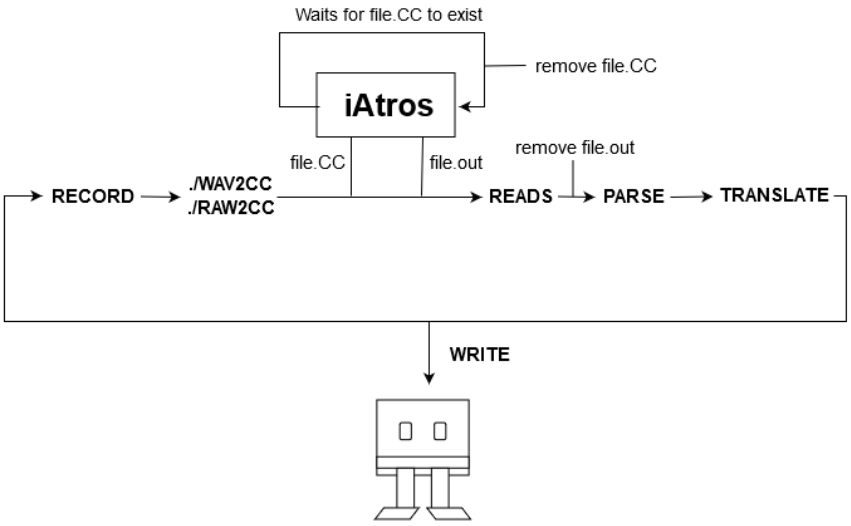

Figure. 17. voiceInterface.py control flow

4. Future work and conclusions

So far, the Automatic Speech Recognition System allows a human-robot directional communication through a console interface which integrates the conceptual principles of the recognition system to command Zowi by voice.

This principle includes the voice command language of the A.S.R.S. and the language that Zowi uses to run and perform its functions. Although the constraint to expand the language according to the expansion capabilities of the device is given by the way the device itself reads the orders and run the callbacks, the general system allows some modification to expand the language in order to make the interaction more natural to the speaker.

4.1. Future work

One possibility is to develop new command semantics as follows in section 4.1.2.

4.1.1 Adding functionalities to the ASRS

For example, to complete the already given possibilities of the functions that Zowi can perform, we can back up to the movement *SerialCommand* previously shown in Figure. 0.3

Particularly, the parameter `MoveSize` is written after the `T` parameter; both are continuous parameters that indicate the size of the movement and the time; respectively. The size of the time is the quantity of time that will take the movement to be performed and the size of the movement is the amplitude of the movements performed by the servomotors. Thus, some semantics can be developed within the A.S.R. language model to accept the order that means a quantity of size that defines those characteristics of movement or how much time the movement will take.

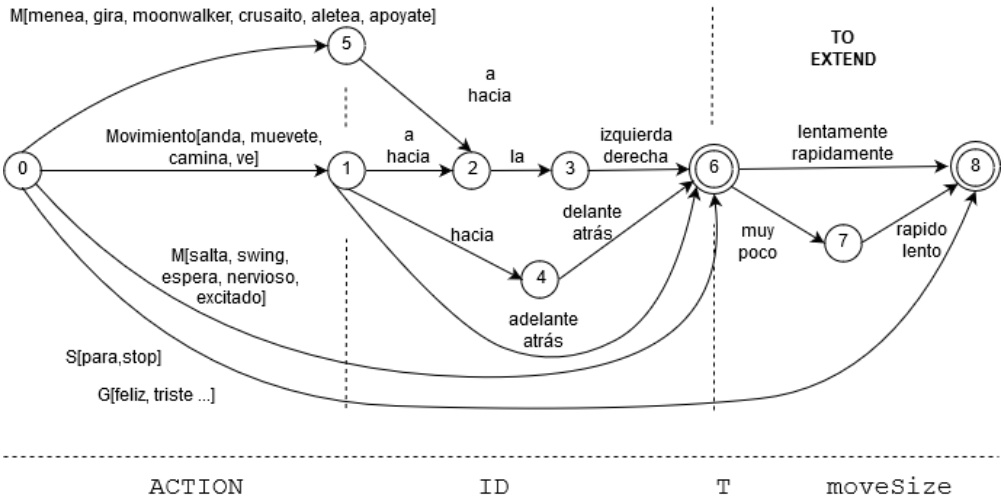

Figure. 18. Extended language model to extend semantics of movement

In this case, as T from time can be given as T, meaning the action is performed an indefinite time, we can extend the language to just change the size of the movement. For that matter, a discrete value range needs to be thought to match the continuous parameter from the semantics of the phrase.

Given the added structure of Figure. 18. as a finite state machine between both possible final nodes, 6 possibilities are contemplated: “muy rapido”, “poco rapido”, “rapidamente”, “lentamente”, “poco lento” and “muy lento”, ordered from their semantics of speed, valued from maximum to the minimum.

Same principles will be applied to the iterative process of the words, where a new modular typed understood as quantity in this case should be defined, where “muy” and “poco” should be placed in *SCmdDICmod* along with the words “rapido”, “rapidamente”, “lentamente,” “lento” and their associated values in *ActmdDIC* of Figure. 06.

In conclusion, enhance the functional capacities of the A.S.R.S. so that the associated action of the device is launched.

4.1.2 Adding functionalities to Zowi

Regarding the device, a future work to considerate is to explore how new functionalities could be applied to Zowi, that is how new actions can be performed if uttered. That would include how to define new robot constants, how to access information from the sensors, and how to make them be requested and called back to be added as a new teleoperation command.

4.2. Considerations

With Zowi, in general, is highly possible to program new functionalities, handle them within the arduino logics, or add them as a *SerialCommand* to be teleoperated. This also means the text processing can be simplified by using a different command prefix to request different functions in the callbacks that mean something similar or displace the logic of the text processing to the robot’s software. At first, the interface can be written directly in C language within the iAtros software to capture the voice and avoid the scripting and the processes integration, which would ease the portability to run the system directly in the embedded device if hardware can manage the requirements; however, bypassing the communication between the devices leads to redefine the device functionality and constrains the computational capacity of the recognition.

4.3. Conclusions

Being able to command by voice an external digital device is not only related to speech recognition itself but to a full extent resolution of the voice recognition application.

To command by voice an external digital device needs some comprehension on how data is treated from the input until the translated action in the robot is done: how the A.S.R. can be tested to improve the performance and which are the possibilities of the recognition with Zowi.

To that end, the applyment of the speech-to-text technology is not only about the theoretical principles of the recognition, but also about the proper integration of the technology into the rest of the components that define the system, guaranteeing that the processes which involve the stages of the project are developed successfully.

5. Bibliography

[1] List of the Bluetooth protocols.
Wikipedia.
https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols

[2] Basics of the UART communication
Web standford class notes
https://web.stanford.edu/class/cs140e/notes/lec4/uart-basics.pdf

[3] Arduino software
Q&A: Can I programm arduino with C?
https://www.arduino.cc/en/Main/FAQ

[4] Finite automata and language models
NLP standford IR book
https://nlp.stanford.edu/IR-book/html/htmledition/finite-automata-and-language-models-1.html

[5] Automatic Speech Recognition definition.
Wikipedia.
https://en.wikipedia.org/wiki/Speech_recognition

[6] Speaker recognition.
Wikipedia.
https://en.wikipedia.org/wiki/Speaker_recognition

[7] An Overview of Speech Recognition Using HMM.
International Journal of Computer Science and Mobile Computing.
http://htk.eng.cam.ac.uk/pdf/woodland_htk35_uea.pdf

[8] Statistical ASR System.
Phil Woodland & Cambridge HTK team: An overview of HTK V3..
http://htk.eng.cam.ac.uk/pdf/woodland_htk35_uea.pdf

[9] Viterbi algorithm.
Wikipedia.
https://en.wikipedia.org/wiki/Viterbi_algorithm

[10] Baud.
Wikipedia.
https://en.wikipedia.org/wiki/Baud

[11] What is RTS/CTS hardware flow control.
Brain boxes.
http://www.brainboxes.com/faq/items/what-is-rts--cts-hardware-flow-control-

Apendix

1 Github repositories documentation of the main board electronics: https://github.com/bq/zowi/blob/master/Zowi_mold/hardware/Zowi%20Main%20Board/PDF/ZOW_Moin_Board.PDF

2 Firmware_v02 of Zowi: https://github.com/bq/zowi/blob/master/Zowi_mold/scr/code/ZOWI_BASE_v2/code%20.hex/ZOWI_BASE_v2.ino

3 Github repositories of the support precompiled libraries: https://github.com/bq/zowi/tree/master/Zowi_mold/scr/arduino%20libraries

4 Github repositories of the Arduino SerialCommand.h library by Steven Cogswell: https://github.com/kroimon/Arduino-SerialCommand/blob/master/SerialCommand.h

5 Pattern recognition of Human Language Technology Research https://www.prhlt.upv.es/wp/

6 pySerial API documentation: https://pyserial.readthedocs.io/en/latest/

7 iAtros by PRHLT: https://www.prhlt.upv.es/software/iatros/doc/offline/

8 recognition Interface [project]: https://github.com/charlDe/recognitionInterface

9 subprocess API documentation: https://docs.python.org/2/library/subprocess.html