# Chapter 4
# Fully Scalable Parallel Hardware for Wheeled Robot Navigation Using Fuzzy Control

**Nadia Nedjah, Paulo Renato S.S. Sandres and Luiza de Macedo Mourelle**

Process control is one of the many applications that took advantage of the fuzzy logic. Controllers are usually embedded into the controller device. This chapter aims at presenting the development of a reconfigurable efficient architecture for fuzzy controllers, suitable for embedding. The architecture is parameterizable so it allows the setup and configuration of the controller, so it can be used for various problem applications. An application of fuzzy controllers was implemented and its cost and performance have been evaluated.

## 4.1 Introduction

Computational system modeling is full of ambiguous situations, wherein the designer cannot decide, with precision, what should be the outcome of the system. In [7], L. Zadeh introduced for the first time the concept of *fuzziness* as opposed to *crispiness* in data sets.

Fuzzy logic and approximate reasoning [6] can be used in system modeling and control as well as data clustering and prediction, to name only few appropriate applications. Furthermore, they can be applied to any discipline such as finance, image

N. Nedjah (✉) · L. de Macedo Mourelle
Department of Electronics Engineering and Telecommunications, State University of Rio de Janeiro, Rio de Janeiro, Brazil
e-mail: nadia@eng.uerj.br

L. de Macedo Mourelle
e-mail: ldmm@eng.uerj.br

P.R.S.S. Sandres
Department of Systems Engineering and Computation, State University of Rio de Janeiro, Rio de Janeiro, Brazil
e-mail: paulo.sandres@eng.uerj.br



N. Nedjah et al. (eds.), *Designing with Computational Intelligence*,
Studies in Computational Intelligence 664,
DOI 10.1007/978-3-319-44735-3_4

processing, temperature and pressure control, robot control, among many others. The fuzzy logic is a subject of great interest in scientific circles, but it is still not commonly used in industry, as it should be. Eventually, we found some literature containing practical applications that is being currently used in industry [3, 4].

There are many related works that implemented a fuzzy controller on a FPGA, but most of them present controller designs that are only suitable for a specific application [2, 4]. Mainly, the designs do not use 32-bit floating-point data. The floating-point data representation is crucial for the sensibility of the controller design. In contrast, all the required computation in the proposed controller are performed by a simple precision floating-point coprocessor.

The purpose of the development of a reconfigurable hardware of a *shell* fuzzy controller, which can include any number of inputs and outputs as well as any number of rules, is the possibility of creating a device that can be used more widely and perhaps spread the concept of fuzzy logic in the industrial final products.

This paper is divided into three sections. First, in Sect. 4.2, we introduce briefly some concepts of fuzzy controller, which will be useful to follow the description of the proposed architecture. Then, in Sect. 4.4, we describe thoroughly, the macro-architecture of the fuzzy controller developed. After that, in Sect. 4.5, we give details about the main components included in the macro-architecture. Subsequently, in Sect. 4.3, we present the fuzzy model used to control the navigational process of a wheeled robot. Then, in Sect. 4.6, we show, via the project of a the fuzzy controller presented, that the proposed architecture is functionally operational and promising in terms of cost and performance. Finally, in Sect. 4.7, we draw some conclusions and point out some new direction for the work in progress.

## 4.2 Fuzzy Controllers

Fuzzy control, which directly uses fuzzy rules, is the most important and common application of the fuzzy theory [5]. Using a procedure originated by E. Mamdani [3], three steps are followed to design a fuzzy controlled machine:

1. fuzzification or encoding: This step in the fuzzy controller is responsible of encoding the crisp measured values of the system parameter into a fuzzy term using the respective membership functions;
2. inference: This step consists of identifying the subset of fuzzy rules that can be fired, i.e., those with antecedent propositions with truth degree not zero, and draw the adequate fuzzy conclusions;
3. defuzzification or decoding: This is the reverse process of fuzzification. It is responsible of decoding a fuzzy variable and compute its crisp value.

The generic architecture of a fuzzy controller is given in Fig. 4.1. The main components of a fuzzy controller consist of a *knowledge repository*, the *encoder* or *fuzzification* unit, the *decoder* or *defuzzification* unit and the *inference engine*. The

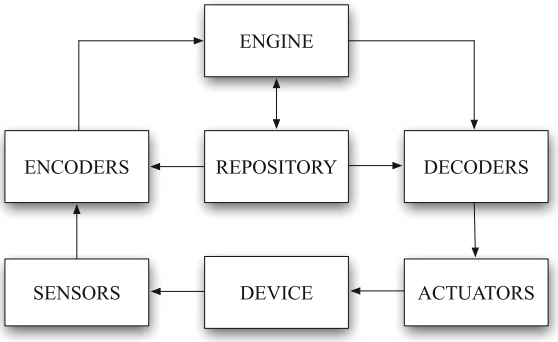



**Fig. 4.1** Generic architecture of fuzzy controllers

knowledge base stores two kind of data: the fuzzy rules that are required by the inference engine to reach the expected results and knowledge about the fuzzy terms together with their respective membership functions as well as information about the universe of discourse of each fuzzy variable manipulated within the controller. The encoder implements the transformation from crisp to fuzzy and the decoder the transformation from fuzzy to crisp. Of course, the inference engine is the main component of the controller architecture. It implements the approximate reasoning process.

## 4.3 Fuzzy Models for Wheeled Robot Navigation

The control of a wheeled robot navigation uses a series of control loops to operate on a surface following a predefined trajectory. Figure 4.2 shows the schematics of the used robot.

This application consists of three subcontrollers: *(i)* the steering control, which uses two controllers requiring two inputs and one output, each; the linear and angular speed controls, which use the same control process requiring two inputs and one output. Although this application has four controllers, this paper will only show one of them, because the drivers are identical in pairs, i.e., the membership functions and rules of the controllers are the same for the the linear and angular velocity.

## 4.4 The Proposed Macro-architecture

The macro-architecture of the proposed fuzzy controller consists of three main units: *(i)* the fuzzification unit (`FU`), which is responsible for translating the input values of the system into fuzzy terms using the respective membership functions. This unit has as many `Fuzzy` blocks as required in fuzzy system model that is being implemented,

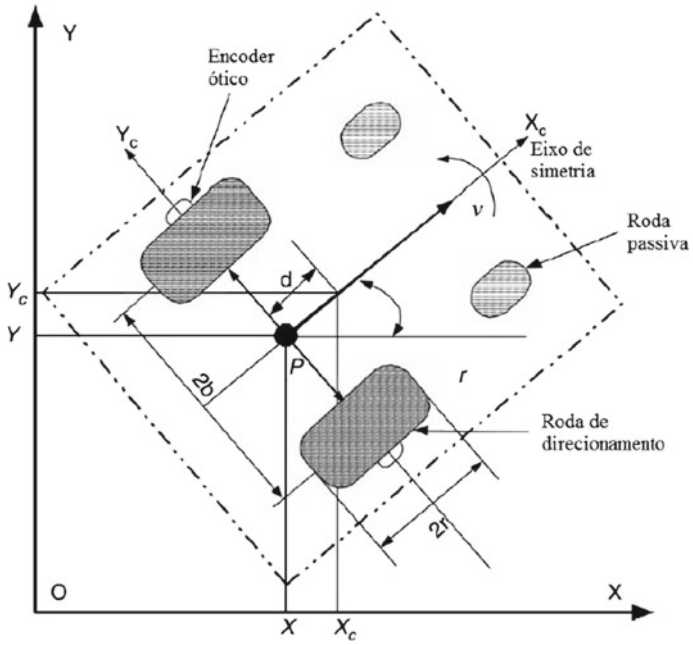



**Fig. 4.2** Model of the wheeled robot used in this application [1]

i.e., one for each input variable; *(ii)* the inference unit `Inference`, which checks all the included fuzzy rules, verifying which membership function applies, and if any is so, generating its value and thus identifying the membership functions to be used in the sequel; *(iii)* the defuzzification unit (`DU`), which is responsible for translating the fuzzy terms back so as to compute the crisp value of the fuzzy controller output. The defuzzification unit includes as many `Defuzzy` blocks as required by the fuzzy system model that is being implemented, i.e., one for each output variable. The block diagram of the proposed macro-architecture is shown in the Fig. 4.3, wherein $N$ and $M$ represent the number of input and output variables, respectively.

Note that, besides the main units, the macro-architecture also includes a component that allows to compute the membership function characteristics, which are used by both the fuzzification and defuzzification units. This component will be called membership function unit (`MFU`). It includes as many `MF` blocks as required input variable of the fuzzy model. Note that all the membership function-related data are stored in the membership function memory, called `MF MEM`. This memory is formed by as many memory segments as required input variables, i.e., one for each membership function used. The rules used by the inference unit are stored in a read-only memory block, called `Rules`. Component `Controller`, which in the sequel may be called *main controller*, imposes the necessary sequencing and/or the simultaneity of the required steps of the fuzzy controller via a concurrent finite state machine. More details on this are given subsequently.

The proposed fuzzy controller is designed to be generic and parametric, so it allows configuring the number of input and output variables, the number of linguistic terms used to model the membership functions, and the number of inference rules, so as the fuzzy system model that is being implemented can fit in. Allowing the configuration of these parameters makes it possible, as well as easy, to adjust the controller design to any desired problem.

As it can be seen in Fig. 4.3, at configuration time, all the membership functions used by the controller are computed and stored in the respective `MF MEM` segment of the membership function memory. All the computed data will be readily available to be used by the pertinent `Fuzzy` and/or `Defuzzy` block in the fuzzification and defuzzification unit, respectively. Note that this configuration step is done only once. During the operation step, the fuzzy controller iterates the required steps, triggering the `Fuzzy` blocks then `Inference` unit then `Defuzzy` blocks in sequence. After that, it waits for a new set of input data to be read by the system sensors and thus arrive at the `Fuzzy` blocks input ports. The finite state machines that control the `Fuzzy` blocks all run in parallel, so do those that control the `Defuzzy` blocks.

In the following sections of this chapter, more light will be shed on the internal micro-architecture of the proposed design as well as the control used therein.

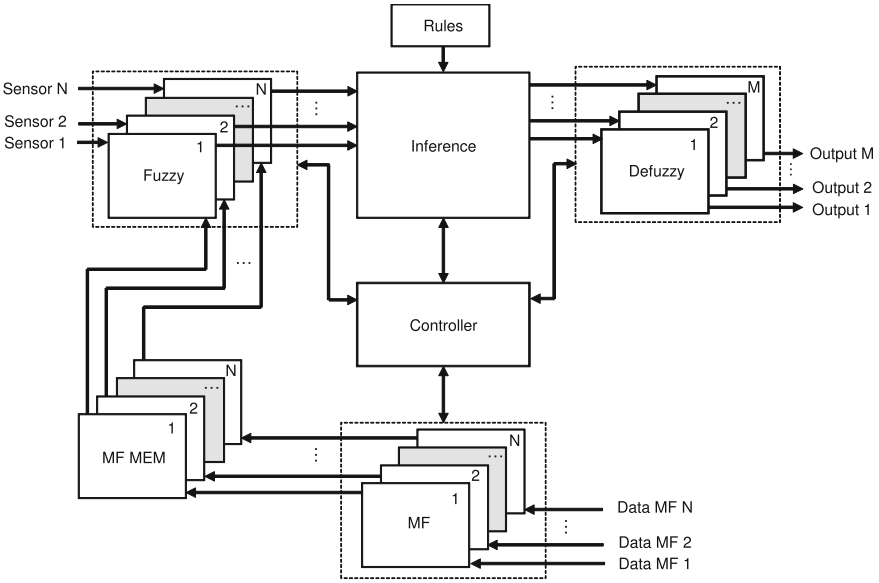



**Fig. 4.3** Macro-architecture of the designed fuzzy controller

## 4.5 Micro-architecture of the Functional Units

In this section, we describe the micro-architecture of the main components, included in the macro-architecture of Fig. 4.3. These are the functional unit responsible for the computation of the member function (`MF`), including the memory-based component (`MF MEM`), the basic component responsible for the fuzzification process (`Fuzzy`), the component that implements the inference process (`Inference`) using the available rule base (`Rules`), and the basic component that handles the defuzzification process (`Defuzzy`). In general, all blocks that perform floating-point computations include an `FPU` unit, which performs the main mathematical operations with simple precision (32 bits). The operations needed are addition, subtraction, multiplication and division.

### *4.5.1 Membership Function Unit*

A membership function is viewed as a set of linguistic terms, each of which is defined by two straight lines. In the proposed design, the triangular shape is used to represent linguistic terms. Nevertheless, it is possible to adjust the design as to accept other used shapes such as trapezes and sigmoid. Figure 4.4 shows a generic example of membership function with $Q$ linguistic terms, wherein the horizontal axis $x$ represents the controller's input, probably read from a sensor, and the vertical axis $y$ represents the truth degree associated with the linguistic terms. This is a real value, between 0 and 1, handled as a simple precision floating-point number of 32 bits. Linguistic terms of triangular membership function are completely defined by $MaxPoint$ or $mp$ and $Range$ or $r$, as illustrated Fig. 4.4.

The `MF` block is designed to compute the values of any variable $x$, according to $y = ax + b$ of the two straight lines, that represent the linguistic term of the membership function. The required basic data that completely define these shapes need to be identified.

The input data of the `MF` block are *MaxPoint* – $Mp$, *Left Interval* – $Li$ and *Right Interval* – $Ri$ for each straight line used to define the linguistic terms of the membership function. The block utilizes them and precompute coefficients $a$ and $b$ accordingly and stored them in the membership function memory segments. Three cases are possible: the leftmost linguistic term (see linguistic term 0 in Fig. 4.4); An in-between linguistic term (see linguistic term 1 and 2 in Fig. 4.4); and finally, the rightmost linguistic term (see linguistic term $Q$ in Fig. 4.4). The computation of $a$ and $b$ of the straight lines of the leftmost, middle, and rightmost linguistic terms are defined as in (4.1)–(4.3), respectively.

$$\mu_L(x) = \begin{cases} 1, & \text{if } x \leq Mp \\ -\frac{1}{Ri} \times x + \frac{Mp}{Ri} + 1, & \text{if } Mp > x \geq Mp + Ri \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

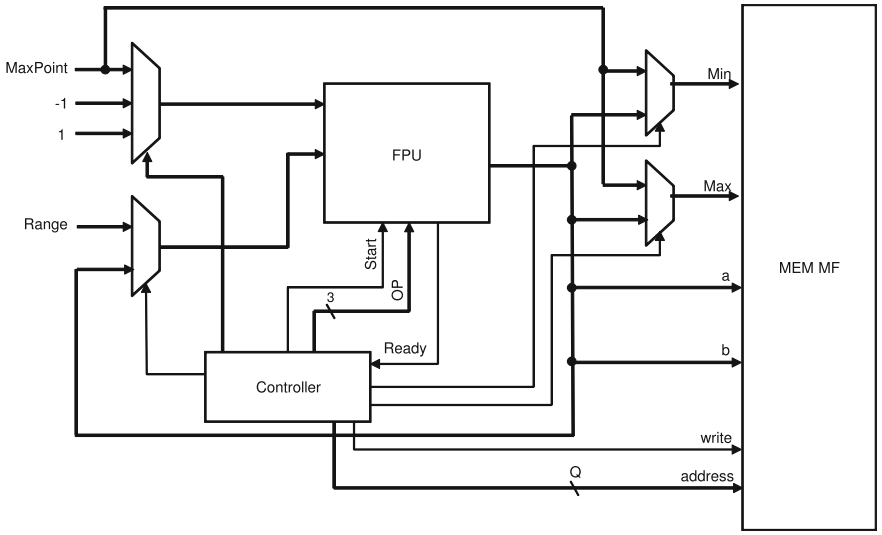



**Fig. 4.4** Membership function of $Q$ linguistic terms

$$\mu_M(x) = \begin{cases} \frac{1}{Li} \times x - \frac{Mp-Li}{Li}, & \text{if } Mp - Li < x \leq Mp \\ -\frac{1}{Ri} \times x + \frac{Mp}{Ri} + 1, & \text{if } Mp > x \geq Mp + Ri \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

$$\mu_R(x) = \begin{cases} \frac{1}{Li} \times x - \frac{Mp-Li}{Li}, & \text{if } Mp - Li < x \leq Mp \\ 1, & \text{if } x > Mp \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

The micro-architecture of the membership function blocks `MF` is shown in Fig. 4.5. It uses a floating-point unit to perform the required mathematical operations. The obtained results are then stored in the `MF MEM` segments.

An `MF` block includes a controller that is implemented as a finite state machine. It allows to synchronize the setting up of all the linguistic terms, necessary to the complete definition of the membership function for each input variable. The control sequence of this controller is given in Algorithm 1.

### *4.5.2 Membership Function Memory*

As explained earlier, this memory block responds to write commands received from the `MF` block and read commands issued by the `FU`. Each word of this memory holds four data that allows the complete computation of the truth degree of a given

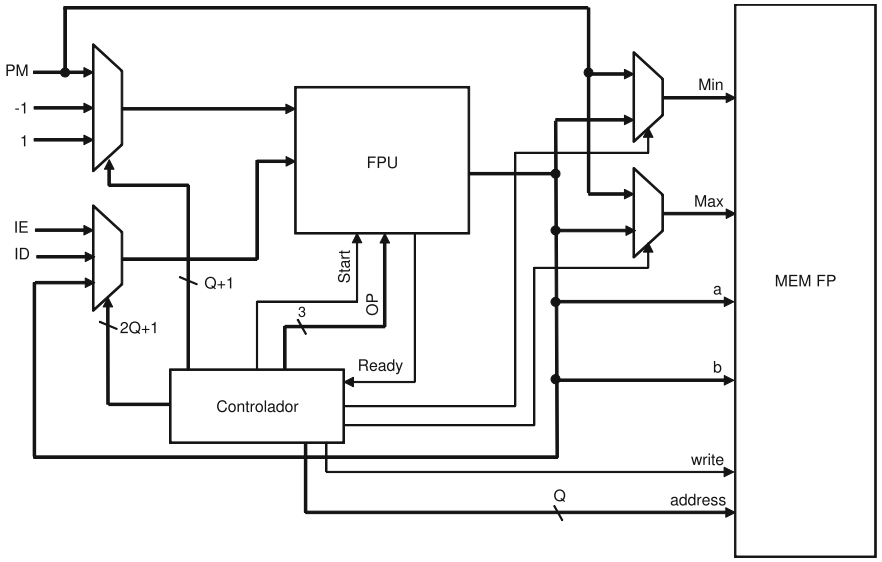



**Fig. 4.5** The micro-architecture of the membership function block

linguistic term. The four-fold memory word contains $min$: minimum limit of the straight line; $max$: maximum limit of the straight line; $a$: angular coefficient of the straight line; $b$: linear coefficient of the straight line.

So, every time the `MF` block requests a memory write, this memory block register these values at an address, that represents the order number of the line within all the line that need to be processed, starting from zero. This block also allows the configuration of the number of lines that can be registered in the memory, which will depend on parameter $Q$, which determines the number of linguistic terms per membership function.

### *4.5.3 Fuzzification Unit*

The `Fuzzy` block performs the necessary computation to obtain the fuzzy version the input value. The computation consists of a comparison that may, in most cases, be followed by a multiplication then an addition, depending on the comparison result. This is repeated $Q$ times for all the linguistic terms included in the membership function of the input variable under consideration. The `Fuzzy` block micro-architecture is shown in Fig. 4.6. It includes a `Comparator` that determines in which linguistic term range the input value falls, 2 sets of $Q$ flip-flops to hold the result of the comparison. Their contents identify which linguistic terms are actually active.

**Algorithm 1** Membership function configuration

**Require:** $Mp_k, Li_k$ and $Ri_k,\ k = 1 \ldots Q$;
**Ensure:** $min_k, max_k, a_k$ e $b_k,\ k = 1 \ldots 2 \times Q$;
**Ensure:** $min, max, a$ and $b$;
**if** $enable = 1$ **then**
  $Address \leftarrow 0$;
  **for** $k \leftarrow 1\ to\ Q$ **do**
    **for** $FP \leftarrow 1\ to\ 2$ **do**
      $Write \leftarrow 0$; $Address \leftarrow Address + 1$;
      **if** $k = 1$ **then**
        **if** $FP = 1$ **then**
          $min \leftarrow -\infty; max \leftarrow Mp_k; a \leftarrow 0; b \leftarrow 1;$
        **else**
          $min \leftarrow Mp_k; max \leftarrow Mp_k + Ri_k;$
          $a \leftarrow -1/Ri_k; b \leftarrow (Mp_k/Ri_k) + 1;$
        **end if**
      **end if**
      **if** $1 < k < Q$ **then**
        **if** $FP = 1$ **then**
          $min \leftarrow Mp_k - Li_k; max \leftarrow Mp_k;$
          $a \leftarrow 1/Li_k; b \leftarrow -((Mp_k - Li_k)/Li_k);$
        **else**
          $min \leftarrow Mp_k; max \leftarrow Mp_k + Ri_k;$
          $a \leftarrow -1/Ri_k; b \leftarrow (Mp_k/Ri_k) + 1;$
        **end if**
      **end if**
      **if** $k = Q$ **then**
        **if** $FP = 1$ **then**
          $min \leftarrow Mp_k - Li_k; max \leftarrow Mp_k;$
          $a \leftarrow 1/Li_k; b \leftarrow -((Mp_k - Li_k)/Li_k);$
        **else**
          $min \leftarrow Mp_k; max \leftarrow +\infty; a \leftarrow 0; b \leftarrow 1;$
        **end if**
      **end if**
      $write \leftarrow 1;$
    **end for**
  **end for**
**end if**

The obtained results for the two straight lines modeling the linguistic term are kept in two distinct 32-bit registers. These are the truth degrees, once it is delivered by the `FPU`. The block includes two sets of 32-bit registers, namely `TuFP1` and `TuFP2`, one for each linguistic term modeling the membership function of the input variable.

The inputs of a `Fuzzy` block are the characteristics of the linguistic terms of the membership function associated with the input variable under consideration. These characteristics are $a$, $b$, $min$ and $max$ stored in `MF MEM` segment corresponding to the input variable, as explained in Sect. 4.5.2. The output of a `Fuzzy` block are: signal $EnF_i$, for $i = 1 \ldots Q$ bits, i.e., one for each included linguistic term and signal $uF_i$,

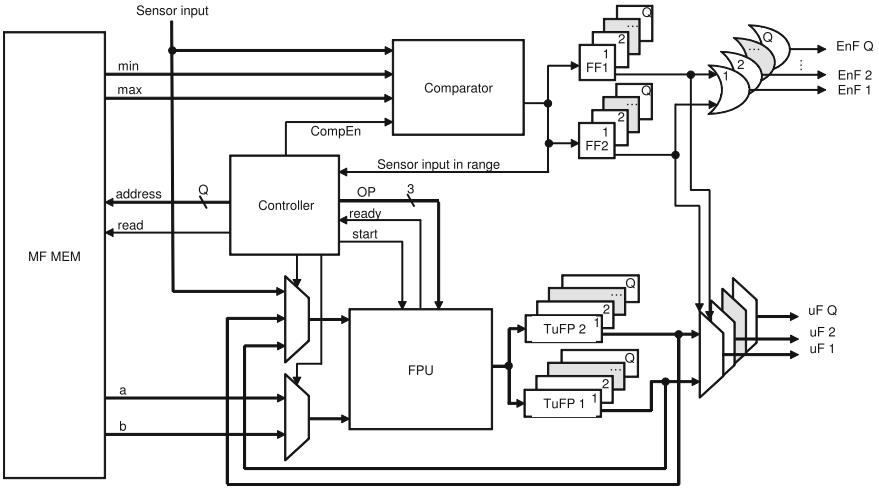



**Fig. 4.6** Fuzzy block micro-architecture

for $i = 1 \ldots Q$ 32-bit floating-point values, each of which represents the truth degree of the corresponding linguistic term. Note that linguistic terms that do not apply have 0 as a truth degree. When bit $EnF_i$ is activated, this indicates that linguistic term number $i$ of the membership function is valid with truth degree $uF_i \neq 0$. Recall that the truth degree is the product of $a$ and input value augmented by $b$. In Algorithm 2, we give an overview on how the `Fuzzy` block operates.

### *4.5.4 Inference Unit*

The inference unit main purpose is to identify, for each one of the output variables of the fuzzy controller, the linguistic terms that are active as well as computing the associated truth degrees.

Before describing the details of the inference unit, let us first introduce the structure used to format the rules of the fuzzy system. A rule $\mathcal{R}$ has two defining parts: a premise $\mathcal{P}$ and a consequent $\mathcal{C}$ as described in (4.4), wherein $\mathcal{I}_i$, for $i = 1 \ldots N$ are the input variables and $\mathcal{T}_k^{\mathcal{I}_i}$ for $k = 1 \ldots Q$ are the linguistic terms associated to it, $\mathcal{O}_j$, for $j = 1 \ldots M$ are the output variables and $\mathcal{T}_k^{\mathcal{O}_j}$ for $\ell = 1 \ldots Q$ are the linguistic terms associated with it. Note that in general, the number of linguistic terms is distinct from one variable to another. However, in this work, we assume, without loss of generality, that all the variables, both of input and output, are modeled using the same number of linguistic terms $Q$. A rule may check only few of of the $N$ input variables, and it may also, enable only few of the output variables.

**Algorithm 2** Operation of the Fuzzification block

**Require:** $sensor, min, max, a$ e $b$;
**Ensure:** $uF_i$ e $EnF_i$, $i = 1 \ldots 2 \times Q$;
**if** $enable = 1$ **then**
**for** $i \leftarrow 1$ *to* $2 \times Q$ **do**
$Address \leftarrow i; read \leftarrow 1;$
**if** $min < sensor < max$ **then**
$FPUout_i \leftarrow sensor \times a + b; COMPout_i \leftarrow 1;$
**else**
$FPUout_i \leftarrow 0; COMPout_i \leftarrow 0;$
**end if**
**end for**
$read \leftarrow 0; k \leftarrow 1;$
**for** $i \leftarrow 1$ *to* $Q$ **do**
$EnF_i \leftarrow (COMPout_k$ OR $COMPout_{k+1});$
**if** $COMPout_k = 1$ **then**
$uF_i \leftarrow FPUout_k;$
**else if** $COMPout_{k+1} = 1$ **then**
$uF_i \leftarrow FPUout_{k+1};$
**else**
$uF_i \leftarrow 0;$
**end if**
$k \leftarrow k + 2;$
**end for**
**end if**

$$\begin{aligned} &\mathcal{R}: \ \mathcal{P} \Rightarrow \mathcal{C}, \text{ where for } j, k, \ell = 0 \ldots Q: \\ &\mathcal{P} \text{ is } \ \mathcal{I}_0 = \mathcal{T}_j^{\mathcal{I}_0} \wedge \mathcal{I}_1 = \mathcal{T}_k^{\mathcal{I}_1} \wedge \cdots \wedge \mathcal{I}_{N-1} = \mathcal{T}_\ell^{\mathcal{I}_{N-1}} \\ &\mathcal{C} \text{ is } \ \mathcal{O}_0 = \mathcal{T}_j^{\mathcal{O}_0} \wedge \mathcal{O}_1 = \mathcal{T}_k^{\mathcal{O}_1} \wedge \cdots \wedge \mathcal{O}_{N-1} = \mathcal{T}_\ell^{\mathcal{O}_{N-1}} \end{aligned} \tag{4.4}$$

The rule base memory `Rules` has a word size that allows to store one rule. All the rules of the model have the same structure. They include all the input and output variables. When a variable is not checked or inferred, the all the linguistic terms are checked off.

A given rule fires when signal $EnF_i$, as delivered by the `FU`, for every checked of linguistic term of every input variable of the premise part of the rule under consideration is set. Furthermore, every linguistic term of any output variable that is checked in the consequent part of a fired rule need to be reported to the defuzzification unit `FU`. Note that there are at most $M$, one for each output variable. Besides this, `FU` needs also to receive the truth degree for each of these checked terms.

The truth degree of an output variable linguistic term is the smallest truth degree, considering all those associated with the input variable linguistic terms in the premise part of the fired rule. When the same output variable linguistic term appears on two or more fired rules, the highest truth degree is used. Thus, this done considering all the rules that fires. Recall that the truth degree of the input variable linguistic terms are provided by the `FU`.

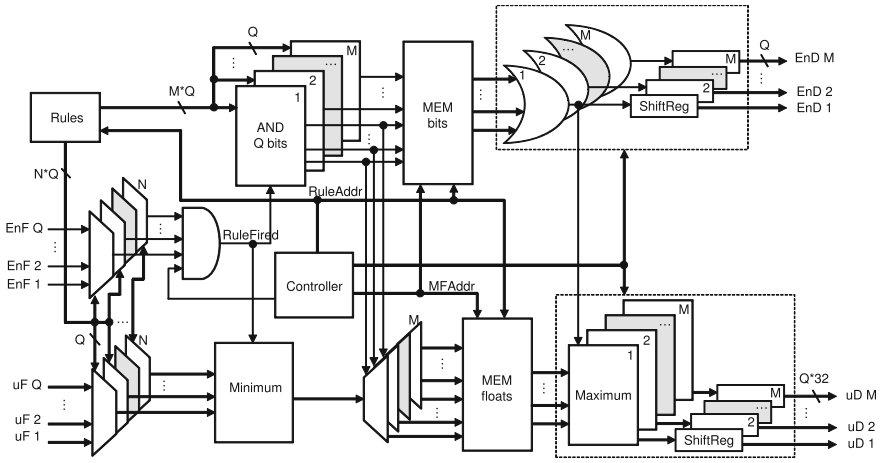



**Fig. 4.7** Inference block micro-architecture

Figure 4.7 shows the micro-architecture of the `Inference` block. Its inputs consist of the $Q$ flags $EnF_i$, for $i = 1 \ldots Q$ and the corresponding $Q$ truth degrees $uF_i$, for $i = 1 \ldots Q$, which are the resulting output of `FU`, as described in Sect. 4.5.3. Its outputs are a set of $M$ $Q$-bit signals $EnD_i$, for $i = 1 \ldots M$, that identify the linguistic terms that were inferred and their respective truth degrees $uD_i$, for $i = 1 \ldots M$, which are signals of $Q \times 32$ bits. The AND gate determines wither the current rule can be fired. In Algorithm 3, we sketch how the operation of the inference block is controlled. The $M$ `ANDQbits` components are simply na AND-arrays. In this design, the process of min-max inference is used. So, components `Minimum` and `Maximum` return the smallest of $N$ floats and the highest of $M$ floats, respectively. Their internal structure is omitted here for a lack of space. The `Inference` includes three memory blocks: the rule base `Rules`, a truth degree memory `MEM floats` and a bit memory `MEM bits`.

### *4.5.5 Defuzzification Unit*

The defuzzification unit's main purpose is to compute the crisp value of the output variables, given the fuzzy linguistic terms and their corresponding truth values, as identified and computed by the inference unit. The centroid is used to perform the defuzzification process. Recall that $uD_i$ for $i = 1 \ldots Q$ are the truth degrees of the linguistic terms associated with the output variable $\mathcal{O}$. The computation is done according to the steps of Algorithm 4.

**Algorithm 3** Inference control and computation

**Require:** $uF_i^j$, $EnF_i^j$, $i = 1 \ldots Q$, $j = 1 \ldots N$ and $Rules_r^l$, $r = 1 \ldots P$, $l = 1 \ldots (N + M) \times Q$;
**Ensure:** $uD_i^k$ e $EnD_i^k$, $k = 1 \ldots M$, $i = 1 \ldots Q$;
**if** $enable = 1$ **then**
**for** $r \leftarrow 1$ *to* $P$ **do**
$\mathcal{R} \leftarrow Rules_r$;
**if** $\mathcal{R}$ Valid **then**
**for** $j \leftarrow 1$ *to* $N$ **do**
$\mathcal{I} \leftarrow \mathcal{R}^j$; $\texttt{AndInput}_j \leftarrow \mathcal{I}$ & $EnF^j$; $\texttt{MinInput}_j \leftarrow uF^j$;
**end for**
$RuleFired \leftarrow \texttt{AND}(AndInput)$;
**if** $RuleFired$ **then**
$Min \leftarrow \texttt{MIN}(MinInput)$;
**for** $k \leftarrow 1$ *to* $M$ **do**
$\mathcal{O} \leftarrow \mathcal{R}^{N+k}$;
**for** $i \leftarrow 1$ *to* $Q$ **do**
**if** $\mathcal{O}_i = 1$ **then**
$\texttt{MEMfloats}_r^{k \times i} \leftarrow Min$; $\texttt{MEMbits}_r^k \leftarrow \mathcal{O}$;
**else**
$\texttt{MEMfloats}_r^{k \times i} \leftarrow 0$; $\texttt{MEMbits}_r^k \leftarrow 0$;
**end if**
**end for**
**end for**
**end if**
**else**
$\texttt{MEMfloats}_r \leftarrow 0$; $\texttt{MEMbits}_r \leftarrow 0$;
**end if**
**end for**
**for** $k \leftarrow 1$ *to* $M$ **do**
**for** $i \leftarrow 1$ *to* $Q$ **do**
$uD_i^k \leftarrow \texttt{MAX}(\texttt{MEMfloats}^{k \times i})$; $EnD_i^k \leftarrow \texttt{OR}(\texttt{MEMbits}^{k \times i})$;
**end for**
**end for**
**end if**

**Algorithm 4** Computation of the centroid

$R_0 \Leftarrow 0$; $R_1 \Leftarrow 0$; $R_2 \Leftarrow 0$;
**if** $EnD \neq 00 \ldots 0$ **then**
**for** $i := 1$ *to* $Q$ **do**
**if** $EnD_i = 1$ **then**
$R_0 \Leftarrow uD_i \times mp_i$;
$R_1 \Leftarrow R_1 + R_0$; $R_2 \Leftarrow R_2 + uD_i$;
**end if**
**end for**
$R_0 \Leftarrow R_1 / R_2$;
**end if return** $R_0$;

## 4.6 Performance Results

The application presented in the following is for the angular velocity control. It requires two input variables that shape the radius and angle in polar form representing the error and error variation of speed, 15 fuzzy rules as described in Table 4.1, 5 linguistic terms and 1 output variable that represents the linear velocity of the robot movement. Figure 4.8 shows the membership functions used for each of the input and output variables.

Table 4.2 shows the sensor input values tested, the rules fired, according to Table 4.1. Also it shows the number of linguistic terms that were activated at the start of the defuzzification process, the number of clock cycles, the execution time in microseconds, based on clock of 112.410 MHz and the scalar value of the result of the hardware controller.

Figure 4.9 shows the control surface based on the configuration of the fuzzy controller for this application. The computation of the quadratic error, as defined in (4.5), is $3.1237 \times 10^{-7}$, which shows an excellent accuracy in comparison to the software implementation using MATLAB. In (4.5), $xh_i$ is the $i$th result returned by the recon-

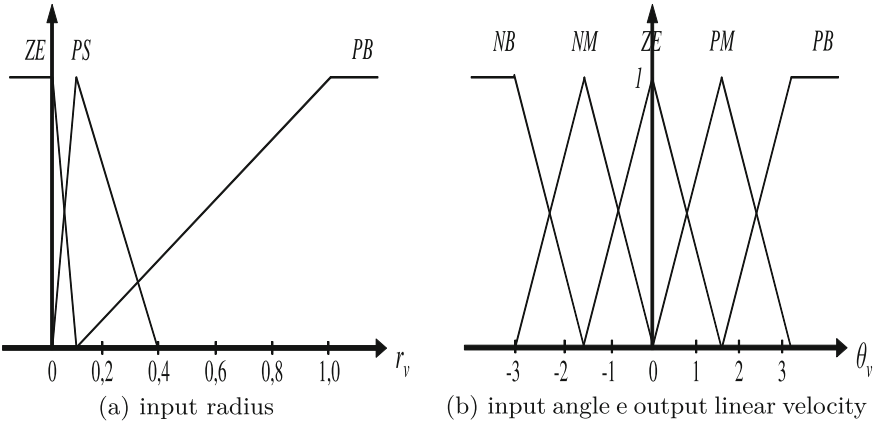


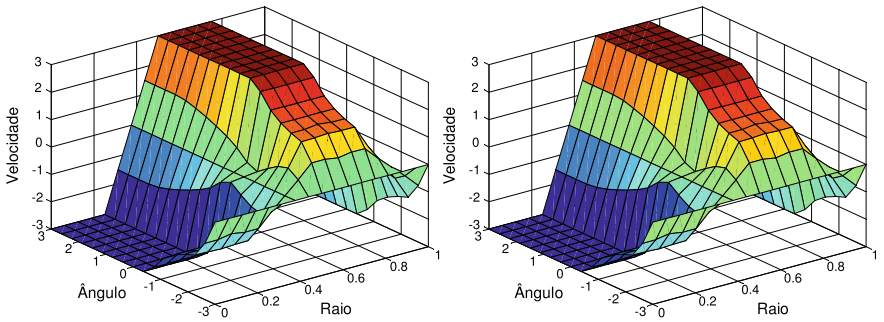

**Fig. 4.8** Membership function used

**Table 4.1** Fuzzy rules for the autonomous robot navigation

| Rule | | Radius | | |
|---|---|---|---|---|
| | | *ZE* | *PS* | *PB* |
| Angle | *PB* | $r_0$: *ZE* | $r_1$: *NM* | $r_2$: *NB* |
| | *PM* | $r_3$: *ZE* | $r_4$: *PM* | $r_5$: *PB* |
| | *ZE* | $r_6$:*ZE* | $r_7$: *PM* | $r_8$: *PB* |
| | *NM* | $r_9$: *ZE* | $r_{10}$: *NM* | $r_{11}$: *NB* |
| | *NB* | $r_{12}$:*ZE* | $r_{13}$: *NM* | $r_{14}$: *NB* |

**Table 4.2** The results obtained by the the reconfigurable hardware for the robot navigation control

| Radius | Angle | Rules fired defuzzy | Number of defuzzy | Cycles clock | Time ($\mu seg$) | Velocity |
|---|---|---|---|---|---|---|
| 0.00 | +1.0 | $r_3$ e $r_6$ | 1 | 1043 | 9.28 | 0.0000 |
| 0.80 | −2.0 | $r_{11}$ e $r_{14}$ | 1 | 1043 | 9.28 | −3.0000 |
| 0.50 | +2.5 | $r_2$ e $r_5$ | 2 | 1090 | 9.70 | −0.4286 |
| 0.05 | −1.0 | $r_6, r_7, r_9$ e $r_{10}$ | 3 | 1137 | 10.11 | −0.1875 |
| 0.09 | +2.0 | $r_0, r_1, r_3$ e $r_4$ | 3 | 1137 | 10.11 | +0.4545 |
| 0.30 | +2.0 | $r_1, r_2, r_4$ e $r_5$ | 4 | 1184 | 10.53 | 0.0000 |
| 0.20 | −0.5 | $r_7, r_9, r_{11}$ e $r_{12}$ | 4 | 1184 | 10.53 | +0.4091 |
| 0.20 | +2.5 | $r_1, r_2, r_4$ e $r_5$ | 4 | 1184 | 10.53 | −0.4091 |




(a) MATLAB (b) Hardware

**Fig. 4.9** Control surface for the wheeled robot navigation

figurable controller hardware, $xm_i$ is the $i$th result of returned by the toolbox FIS of MATLAB, and $n$ represents the total number of obtained results. In this case, we use 17 distinct set of inputs.

$$Error = \frac{\sum_{i=1}^{n}(xh_i - xm_i)^2}{n} \tag{4.5}$$

Using a clock frequency of 100 MHz in FPGA, the entire controller runs, in the worst case, with Defuzzy4 in 2,246 clock cycles, i.e., 22.46 µs. The synthesis results show that the maximum clock frequency accepted by the design developed for this application is 112.410 MHz, which resulted in an execution time of 19.98 µs. As the operation time of block `FP` is not accounted for in the normal cycle of the controller loop, the latter will be at most of of 1,184 clock cycles, i.e., 10.53 µs, considering the maximum allowable clock frequency. Figure 4.10 displays the number of clock cycles for each block of reconfigurable controller, including variations in numbers of cycles for block `Fuzzy`. Figure 4.11 shows the execution times of each block, using the maximum clock frequency.

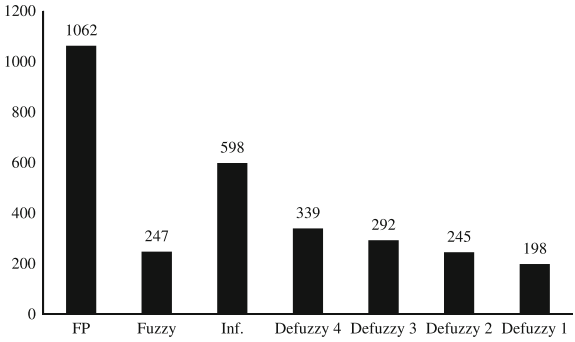



**Fig. 4.10** Number of clock cycles required by the reconfigurable controller

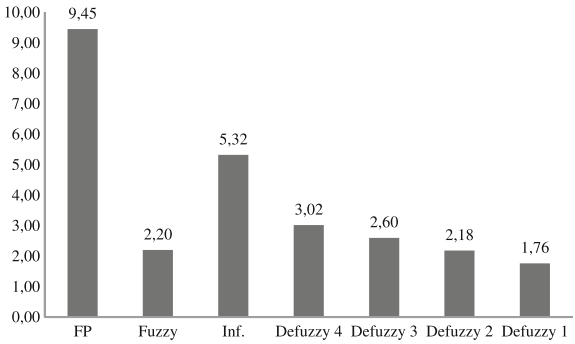



**Fig. 4.11** Execution time, in microseconds, of the blocks in the FPGA

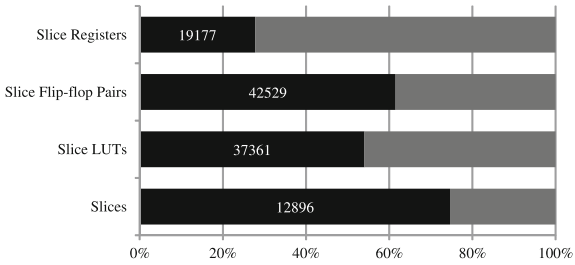



**Fig. 4.12** Hardware area usage in the FPGA

Figure 4.12 shows the hardware area required in the FPGA to program the entire fuzzy controller. Considering the 69,120 LuTs available in the FPGA, only 54.1 % and used.

## 4.7 Conclusion

This paper proposes a massively parallel completely configurable design for fuzzy controllers. It is applicable to almost any applications in the industry that do not have a prescribed solution. The proposed architecture is parametric so that any number of

inputs, outputs, and rules can be accommodate with no extra effort. The design was implemented on reconfigurable FPGA and the cost and performance requirements analyzed. The fuzzy controller supervises the navigational process of a wheeled robot. The next steps in the design of this controller are to investigate the generalization of the design so that to allow the use of trapezoidal and sigmoid the membership functions.

## References

1. Lin WS, Huang CL, Chuang MK (2005) Hierarchical fuzzy control for autonomous navigation of wheeled robots. IEE Proc-Control Theory Appl 152(5):598–606
2. McKenna M, Wilamowski B (2001) Implementing a fuzzy system on a field programmable gate array fuzzy sets and systems. In: Proceedings IJCNN '01 - international joint conference on neural networks, vol 1, pp 189–194. IEEE, Washington, DC
3. Pappis CP, Mamdani EH (1977) A fuzzy logic controller for a traffic junction. IEEE Trans Man Cybern 7(10):707–717
4. Poorani S, Urmila Priya T, Udaya K, Renganarayanan S (2005) FPGA based fuzzy logic controllers for electric vehicle. J Inst Eng 45(5):1–14
5. Zadeh L (1968) Fuzzy algorithms. J Inf Control 12(2):94–102
6. Zadeh L (1984) Making computers think like people. IEEE Spectr 21(8):26–32
7. Zadeh L (1988) Fuzzy logic. IEEE Comput J 21(4):83–93