Proceedings of the International MultiConference of Engineers and Computer Scientists 2018 Vol II
IMECS 2018, March 14-16, 2018, Hong Kong

Deployment of a Low Cost Fuzzy Controller Using Open Source Embedded Hardware and Software Tools

Ronald A. Ponguillo, *Member, IAENG*

***Abstract*— This paper describes implementation of a Fuzzy Logic controller into a Raspberry Pi 2B+ platform. The development was made using Octave software and C++ language for implement the mathematical model for the controller. The Linux distribution used was Raspbian. Several tests were performed to try out how good can be a control system mounted over low cost platform. The tests performed consisted on measure the response of the controller when the reference is fixes and when it is variable. The run times for the algorithm implemented and the CPU consumption form the system were measured.**

The tests results shows that is possible implement this type of control using this approach, but raises a question to answer. It is possible to implement whatever kind of controller from its mathematical model using low cost embedded platform? The experiments showed when the controller is most sophisticated the computational cost grows. The time for initialization resulted bigger than others types of simplest controllers but to the end the control was possible.

***Index Terms*— Embedded automatic control, Octave, Fuzzy Control, Raspberry Pi B2+. Linux**

I. Introduction

In automatic control, there are several options to build the controller for a system. It can be from classical feedback control, PID control, Fuzzy Logic control, Neural Network control, combination of them and others most advanced.

The most popular controller for its mathematical simplicity and good performance is called PID (Proportional, Integral, and Derivative) [6]. For many applications, the PID controller is enough, but in others is necessary to improve the performance of controller. One problem of PID controller is the big over shutting and large stabilization time.

This project implements a Fuzzy Logic controller as an alternative control for processes. A fuzzy controller produce a response faster than PID and follows the reference with greater fidelity when it is fixed as well as when is variable.

The plant to control is the same as in [1]. A two coupled DC motors, one is the plant and the other is the sensor that feedback the input. The procedures to make the data acquisition and test the system are described in [1].

For implement the fuzzy logic controller [2] is necessary to analyze the plant, write the membership functions and test the system to validate the model.

II. Procedures

A. Fuzzy Model

For begin to specify the algorithm used inside of the fuzzy control, first should determine the membership functions for each control variables. As input it has, the voltage delivered for the generator in continuous way and feedback to the system. Also, was defined the PWM signal as the output of the system in which is controlled the duty cycle.

Although the input signal received from the sensor is a voltage, in the membership function was used the error of this voltage when it is compared with the reference signal, such as showed in the figure 1.

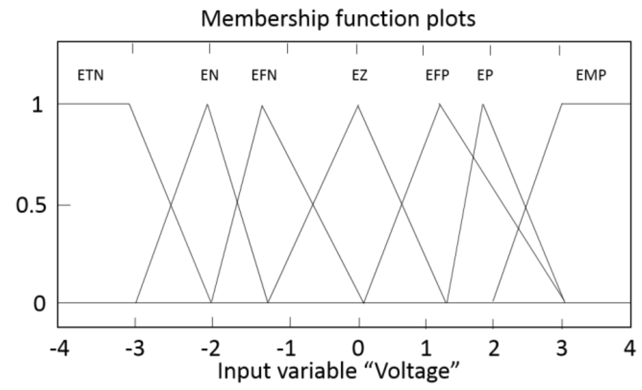

Fig. 1. Membership function for the voltage error

Where the proposed membership functions for the input voltage are:

EZ : Error Zero
EFP : Error Few Positive
EP : Error Positive
EMP :Error Too Positive
EFN : Error Few Negative
EN : Error Negative
ETN : Error Too Negative

Respect to the output, the membership functions showed in the Fig. displays seven levels for linguistics labels for the duty cycle.

Manuscript received May 8, 2017; revised May 10, 2017.

R. A. Ponguillo is the Coordinator of the Basic Electronics Area and Researcher of the Vision and Robotics Center at the Escuela Superior Politécnica del Litoral, ESPOL, Faculty of Electrical and Computer Engineering, Campus Gustavo Galindo, Km 30.5 Via Perimetral, P.O. Box 09-01-5863, Guayaquil, Ecuador (e-mail: rponguil@espol.edu.ec).

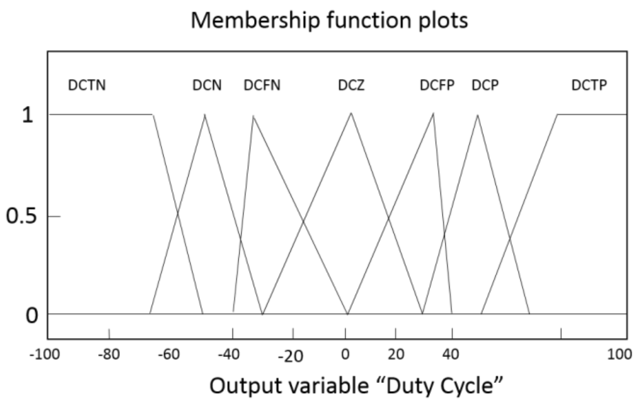

Fig. 2. Membership function for duty cycle

The membership functions for the duty cycle are showed below:

DCZ : Duty Cycle Zero
DCFP : Duty Cycle Few Positive
DCP : Duty Cycle Positive
DCTP : Duty Cycle Too Positive
DCFN : Duty Cycle Few Negative
DCN : Duty Cycle Negative
DCTN : Duty Cycle Too Negative

B. *Algorithm for Fuzzy Control Generation*

In the figure 3 is presented the algorithm implemented for the code generation process. Unlike PID controller, in fuzzy controller is necessary to define fuzzification and defuzzification processes. Fuzzification process refers to the extraction on the membership function, the index that represent the input value, for this case, the voltage error. On the other hand, defuzification process imply to convert the fuzzy sets to a numeric value for generate the control signal, in this work is the PWM duty cycle. There are several methods for defuzzification of a data set: centroid, middle of maximum (MOM), last of maximum (LOM), among others. In this project was used to the centroid method.

C. *Observations in Code Programming*

The code written for run in the Raspberry Pi [5] was written in C++ language and use dependencies for to do interface with Octave and use functions and toolbox for automatic control of Octave.

To write the code for run fuzzy control over Octave is necessary take some considerations. First, it should initialize the libraries to use Octave functionalities and peripherals for interact with the external environment. The lines of code bellow shows the dependencies that must be imported.

```
#include <iostream>
#include <octave/oct.h>
#include <octave/octave.h>
#include <octave/parse.h>
#include <octave/toplev.h>
#include <math.h>
extern "C"{ #include "perifericos.h"}
```

As a second step is necessary to write the additional functions prototypes, created for rounding, round and fuzzification.

```
int rounding(float);
int fuzzification (Matrix,float);
float round(double);
```

The most important function created is fuzzification( ). With this function, error data is taken and search inside the matrix, the value to the index where this data is inside of membership function, and then this data is converted in a fuzzy data. The other two functions round out the values making this in float rounding and the second one obtain the upper maximum for an integer number.

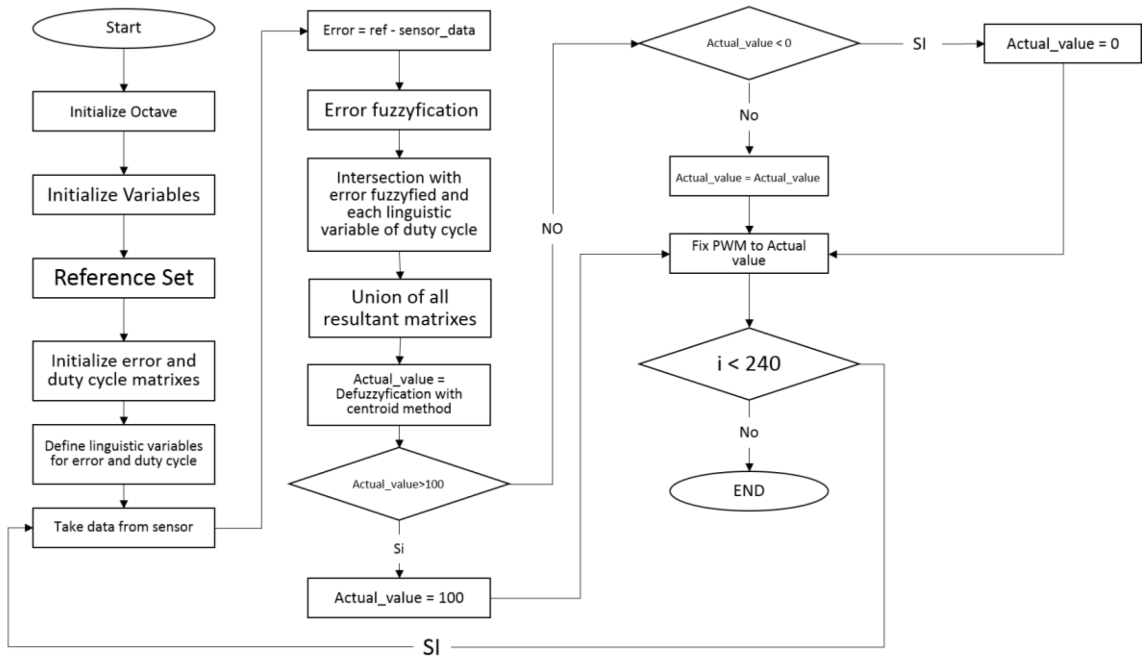
Start
Initialize Octave
Initialize Variables
Reference Set
Initialize error and duty cycle matrixes
Define linguistic variables for error and duty cycle
Take data from sensor
Error = ref - sensor_data
Error fuzzyfication
Intersection with error fuzzyfied and each linguistic variable of duty cycle
Union of all resultant matrixes
Actual_value = Defuzzyfication with centroid method
Actual_value>100
Si
Actual_value = 100
NO
Actual_value < 0
SI
Actual_value = 0
No
Actual_value = Actual_value
Fix PWM to Actual value
i < 240
No
END
SI

Fig. 3. Flowchart for fuzzy control algorithm

```
int fuzzification( Matrix e, float error)
{ int i=0;
   int n=0;
   for(i=0;i<801;i++)
   {  if(error == (float)e(0,i))
      {    n=i;
         break;  }
   } return n; }

float round (double value)
{ float tmp = round((double)value*100.0);
   tmp = tmp/100.0;
   return tmp; }

int rounding (float value)
{ int result=0;
   if ((value - floor(value)) <= 0.5)
      result= floor(value);
   else result = floor(value) + 1;
   return result; }
```

In the same way as libraries were instantiated, is necessary to embed the Octave interpreter inside of C++, so the compiled code can make use of the Octave functionalities. It is very important write the following lines in the main function.

```
string_vector argv (2);
argv(0) = "embedded";
argv(1) = "-q";
octave_main (2, argv.c_str_vec (), 1);
```

The next step is to define the variables program, specify the channel for ADC (Analog to Digital Converter) and run this and the PWM module. At the end, was defined all matrixes for the error values and duty cycle.

When membership functions are executed, the values obtained must be saved for the subsequent defuzzification process. These were place in for loop that takes the next data from the feedback, as well as the value of the subtraction between this feedback and the reference for obtain the error signal. This error signal is fuzzyfied to get the fuzzy dataset. The following step is to evaluate each error value in the error membership function and in its corresponding membership function for duty cycle. Then the intersection of fuzzy sets was made and the minimum values were taken, i.e. the values that are below the fuzzy error.

Once it has the result with fuzzy error, must do the union of all sets of membership functions for the output, using the theory of maximum possible values [2]. By taking these values, was done the defuzzification process applying the centroid method [2]. This consists of taking all the resulting data from the union as if it were a mass function and calculating the center of mass. The result of this defuzzification will be the new setting value.

```
defuzzy(0)=cycle_duty;
defuzzy(1)=b(0).matrix_value();
defuzzy(2)= "centroid";
C_duty=feval("defuzz", defuzzy,3);
result = C_duty(0).float_value();
```

With these new setting values, the next step in the controller design is the same as PID classical controller made in [1]. This setting value will be added before. While error is bigger, the new setting also, however the time of stabilization is too faster. The round function was used because the result obtained in the output is a decimal number with fraction and the value that should be loaded in the PWM module must be integer.

Like to PID controller, values must be saturated from 0 to 100.

```
actual_value = actual_value+result;
if (actual_value > 100)
   actual_value =100;
else if (actual_value < 0)
   actual_value =0;
pwm_value(rounding(actual_value));
```

It is strong recommended, after the process is completely executed, to erase the buffers using the next code.

```
pwm_value(0);
clean_up_and_exit (0);
```

III. TEST AND RESULTS

The evaluation of effectiveness of software tests must be consider two aspects: efficiency tests and sufficiency tests. The efficiency tests involve: performance, run-time and memory management tests, while the sufficiency tests refer to coverage tests.

Embedded systems generally are systems that works in real time. For this reason, the time response is a very important factor when it is spoken about the quality of the product. In fact, software developers need to make performance tests to ensure efficient software.

On the other hand, embedded systems have less resources than general purpose computers. Associated to performance are the memory management and consumption of CPU processing capacity.

A. Performance Test

For performance test, it is needed to measure the run-time for the system and the time that take executes the C++ code written. This C++ code also perform the interface function between the peripherals and the functions developed in Octave.

Run-time Test on C++

Function clock_gettime

This function provides the system time. With this can take the time when an event start and ends. Finding the difference between these two values it is possible to determine the initialization time and run-time for each iteration in the program.

Testing memory consumption and CPU utilization

Sysbench tool in Linux

Embedded Systems are characterized by its limited amount of available memory. For this reason, a good embedded software must have an adequate use of memory. Sysbench tool are a set of libraries that enable to make some specifics testing in a Linux system [4]. Among the tests carried out by this library:

- Calculate the first 5000 prime numbers
- Reading and writing data transfer rate

TOP Tool in Linux

Typically, operating systems have tools for measuring and controlling the processes they are running. In Linux, the TOP tool is the one that performs this function, since it provides all the necessary information in terms of CPU utilization, run- time, memory usage among other data.

In this case, the use of this tool allows read the information about the percentage of CPU consumption for the Fuzzy controller.

Run-time test of Fuzzy controller

For test the Fuzzy controller in the Raspberry Pi board it was developed the next steps:

1) Connect the Raspberry Pi 2B+ board to the plant.
2) Connect the RJ45 cable among Raspberry board and network connection.
3) Turn on the system.
4) Open a terminal with SSH protocol in a PC, and connect with Raspberry Pi board using the IP address: 200.126.1.160:22.
5) In PC terminal enter user: pi and password: raspberry. If all is right should appear the prompt **raspberrypi#**
6) When the terminal session is open, find the folder /**raspberry_c++** and run the commands for compile the Fuzzy controller:
   raspberrypi# mkoctfile --link-stand-alone -lrt -lpthread -lpigpio control_fuzzy_mediciontiempo.cc perifericos.o -o control_fuzzy_mediciontiempo
7) When the compilation process is finished, run the controller using the comand:
   raspberrypi#sudo ./control_fuzzy_mediciontiempo

CPU consumption test for Fuzzy controller

To obtain the CPU consumption, the following steps were taken:

1) Connect the Raspberry Pi 2B+ board to the plant.
2) Connect the RJ45 cable among Raspberry board and network connection.
3) Turn on the system.
4) Open a terminal with SSH protocol in a PC, and connect with Raspberry Pi board using the IP address: 200.126.1.160:22.
5) In PC terminal enter user: pi and password: raspberry. If all is right should appear the prompt **raspberrypi#**
6) When the terminal session is open, find the folder /**raspberry_c++** and run the commands for compile the Fuzzy controller:
   raspberry# mkoctfile --link-stand-alone -lrt -lpthread -lpigpio control_fuzzy.cc perifericos.o -o control_fuzzy
7) Before to run the controller, in the PC, open another console with SSH connection such as step 4. Run the command:
   raspberry# top –d 0.5 –b >> reporte_completo.txt
8) Step 7 started the acquisition data for CPU processes and saving in file reporte_completo.txt. Now in the first SSH terminal should be run:
   raspberry# ./control_fuzzy
9) Once the execution of the controller is finished, the TOP tool is also stopped top. In the session opened in step 7 it ends with Ctrl+C keys combination.
10) The file obtained in step 7 contains all processes that runs in the system. It is important filter data for the process control_fuzzy only. The command used was:
   raspberry# cat reporte_completo.txt | grep control_fuzzy >> reporte_cpu.txt

CPU, File Reading and Writing Benchmark using Sysbench Tool

As mentioned before, Sysbench is a tool that allows quantify the capacity and performance of Linux systems [4]. Test perform in this stage consist of the next steps:

1) Open a SSH session in a PC, then in the console must execute the command for install sysbench:
   # sudo apt-get install sysbench
2) Once the program was installed, type the command to create a process for calculate the first 5000 prime numbers.
   # sudo sysbench --test=cpu --cpu-max-prime=5000 run
3) When the step 2 is finished, it is displayed a complete report that shows the details of execution.

For calculate the time to read and write in the memory, the following steps are executed:

1) Prepare the files for data transfer, typping:
   # sysbench --test=fileio --file-total-size=2G prepare
2) After that files are created, it started the data transfer from SDCard running the command:
   # sysbench --test=fileio --file-total-size=2G --file-test-mode=rndrw --init-rng=on --max-time=300 --max-requests=0 run
3) At the end, when the data transfer is completed, a report with the transfer rates and another relevant information of the process is generated.

B. Results for Fuzzy Control Test

The physical connections for the hardware tests, were implemented like to [1], using the Diligent acquisition system and the same plant.

It can be seen in Fig. , the result of Fuzzy control has less overshoot and ripple compared with PID controller in [1]. On the other hand, with fuzzy is required greater impulse for go from zero to the reference value, but when the error decrease, the fuzzy controller follow better its reference than the PID in [1].

Something that can observe is the time for convergence of the control algorithm. For the PID in [1] the stabilization time is around of 20 seconds while for it new approach is near to 100 seconds. Really, Octave like Matlab uses matrix for its calculations. In the PID controller, the constants for proportional, derivative and integral parts for the controller are calculated previously. Then for each iteration is figured out the error output and its products with every constant of the controller. In the Fuzzy controller, rules for fuzzification and defuzzification processes employ a large number of iterations for develop operations among matrixes, which involve a higher computational cost and can be observed as demand of five times the initialization time compared with [1].

To verify how good the controller is, other experiment was made. In figure 5 can see the response of controller when the reference is varied from three volts stabilized as fixed standard value to down and up from this value. The results show the time for initializing and stabilized the controller is near to 100 seconds. Then the fuzzy controller follow the reference signal in each change without increase the delay or ripple.

C. Results for Software Test

The response of the whole system is determined for two parameters: one is the quality and performance of hardware and the other one is the performance of the software that runs in this hardware. To validate the performance of software, four test was developed. The time for each iteration in the program was taken. It was considered also the peripheral instantiation and initialization of the linguistics variables. Others considerations were the time for error calculate, to make fuzzification and defuzzification processes and calculate and send the new PWM value. The complete values for the whole process are shown in Table I.

Per the data in Table I, it can observe that the average time for iterations in Fuzzy controller is almost 8 times than in the PID alone [1], 1.9144 vs 0.2432 seconds.

D. Results for CPU Use Test

In fig. 6, it can see the use of CPU in the raspberry Pi board while the fuzzy controller is running. It can notice that initially use is around 21% of total ARM Quad Core Processor before to run the fuzzy control script. Once the control script is running this value up to 28% peak. This is due to initialization of peripherals, variables and environment. After, this consumption decrease to 26.7% while linguistics variables are initializing and continue decreasing to 24.9% when takes the data and calculate the new value for PWM. This results was measured with TOP tool, which allow view each process that runs in the Linux system [4].

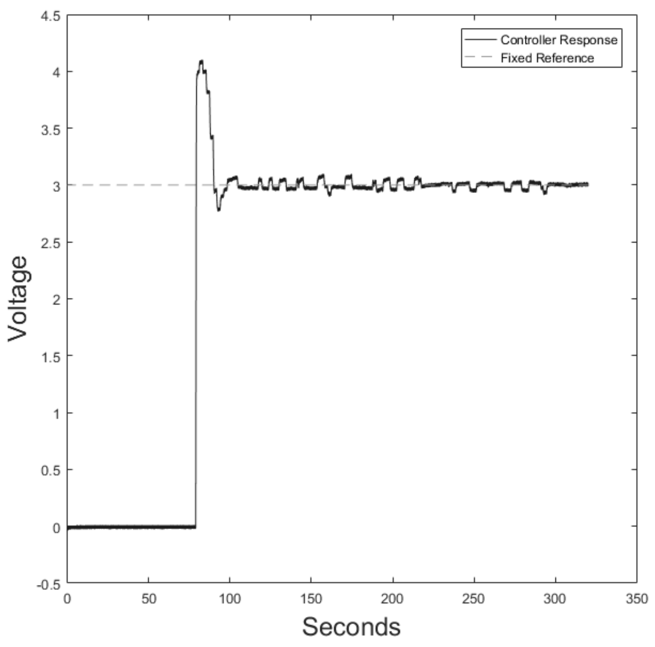

Fig. 4. Response taken with a fixed reference in Fuzzy controller

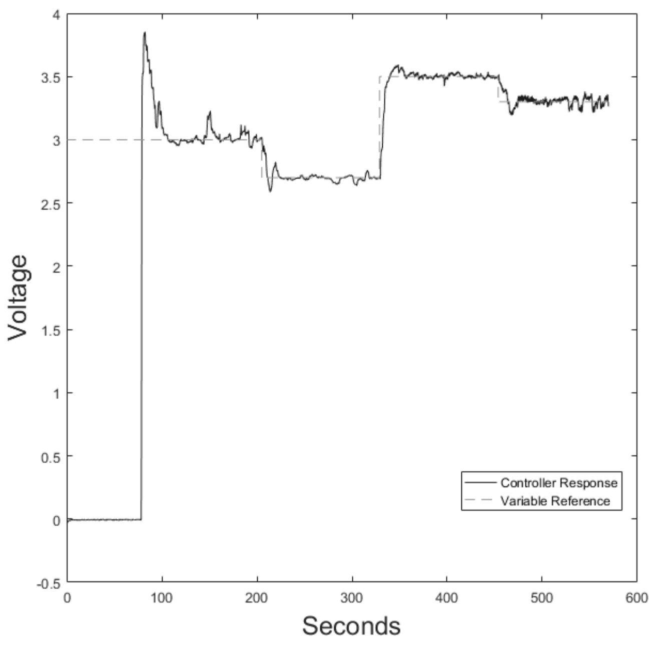

Fig. 5. Response for Fuzzy controller when reference is variable

TABLE I
RUNTIMES FUZZY CONTROLLER IN SECONDS

Test	Initialization	Sum of Iterations	Average Iterations	Slower Iteration	Total time
T1	78.7138	463.97	1.9392	2.1503	542.6838
T2	76.2865	459.35	1.9140	2.0714	535.6365
T3	74.6023	448.1800	1.8674	1.9323	522.7823
T4	75.5927	464.8600	1.9369	2.0996	540.4527
Avg.	**76.2988**	**459.090**	**1.9144**	**2.0634**	**535.3888**

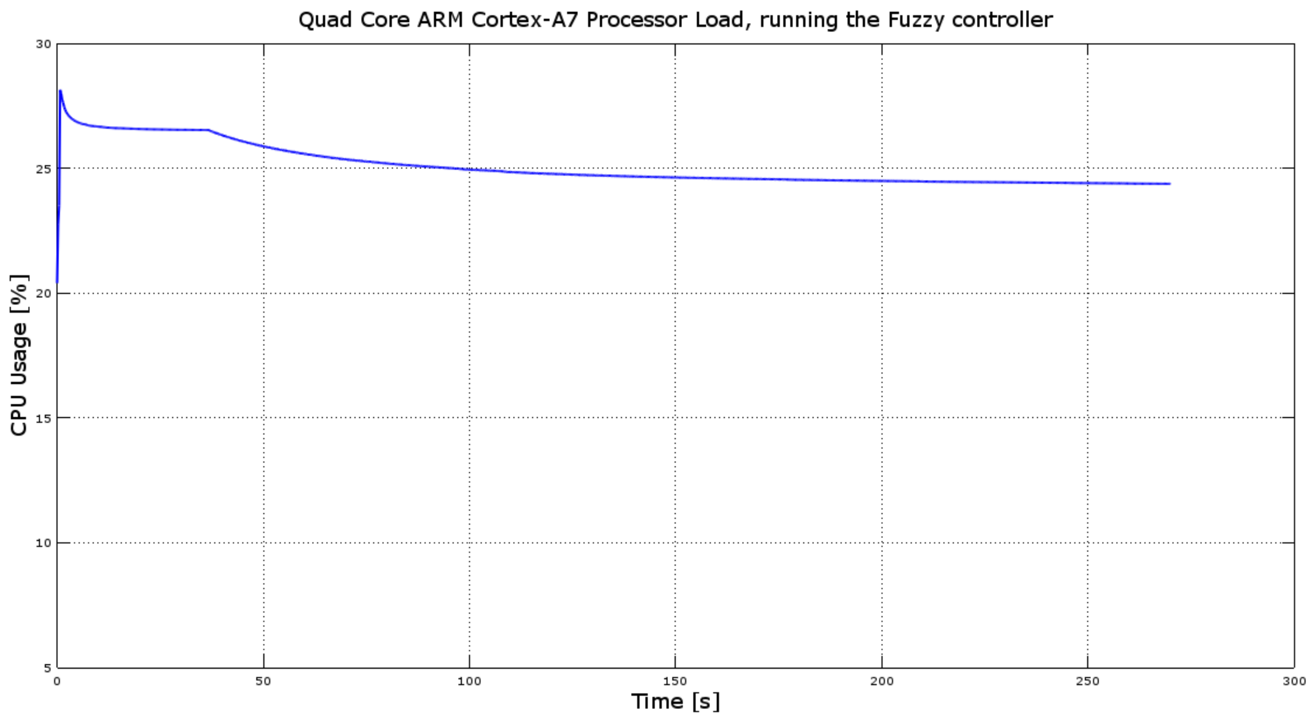

Fig. 6. Percentage of CPU usage while the Raspberry Pi executes PID Fuzzy controller

IV. CONCLUSION

This paper presented a Fuzzy Logic control on embedded system based on low cost platform Raspberry Pi B2+ suggested in [1]. The model for Fuzzy controller was similar as [3]. The results show the Fuzzy control is possible with proposed technique but the computational cost was bigger than the previous work. The system was stabilized but both the time for initialization and number of iterations for control were bigger. For systems with faster dynamics requirements it is possible the proposed embedded platform not work.

As a future work, it is proposed to work to improve embedded system performance using clustering computing techniques and parallel programing in Octave.

REFERENCES

[1] Ponguillo, R.A. and Medina, C.V., “Using Open Source Embedded Hardware and Software Tools in Automatic Control from Mathematical Model,” in Lecture Notes in Engineering and Computer Science: World Congress on Engineering and Computer Science 2016, pp. 333-337.
[2] PASSINO, Kevin M.; YURKOVICH, Stephen; REINFRANK, Michael. Fuzzy control. Reading, MA: Addison-Wesley, 1998.
[3] Cadena, A., Ponguillo, R., & Ochoa, D. (2017). Development of Guidance, Navigation and Control System Using FPGA Technology for an UAV Tricopter. In Mechatronics and Robotics Engineering for Advanced and Intelligent Manufacturing (pp. 363-375). Springer International Publishing.
[4] MARSH, Nicholas. Introduction to the Command Line: The Fat Free Guide to UNIX and Linux Commands. CreateSpace, 2010.
[5] Upton, E., & Halfacree, G. (2016). Raspberry Pi User Guide.
[6] Ogata, K., & Yang, Y. (1970). Modern control engineering.