# Embedding-code for a Mobile Fuzzy Logic Controller

G. N. Reddy, Gurpreet Singh, Vishnudev Vasanthan
Drayer Department of Electrical Engineering
Lamar University
Beaumont, TX, USA
gnreddy@lamar.edu

***ABSTRACT:*** *This paper presents embedding-code that can be used to realize a mobile fuzzy logic controller FLC. The software code is developed using MS Visual C++. The code can be downloaded into any microcontroller board MCB that supports C++. A short review of the current MCBs is included. Some mobile-FLC-applications are cited. The software-code is an FLC-simulator. It can be used to simulate an FLC off-line or it can be downloaded into a mobile-microcontroller so it can operate in autonomously. The embedding-code has 4-modules, 3-input and 1-output. The three input modules represent system description in the form: of a rule-based knowledge base; a set of input/output variables with their corresponding membership functions; and a set of sensed input variable values. The output module contains the generated control signals. It also generates continuous step-by-step output simulation trace illustrating the detailed sequential steps executed by the FLC's inference engine. While operating off-line, the three input modules are represented by three input data files and the output module by a generated output file. While operating in real-time, the knowledge-base and the i/o membership functions are merged into the FLC's C++ code. The sensed input signals and the computed output control signals are mapped to the target microcontroller's i/o-ports. In real-time mode the output simulation trace can be disabled. We have used this software to implement two RT-FLCs: One for precise estimation of the state of charge SoC of a battery using what is called the impedance interrogation method; and the other to implement the classical controller for balancing of an inverted pendulum.*

**Keywords:** Fuzzy Logic Expert Systems FLES, Real-time Fuzzy Logic Controllers RT-FLC, Embedding-code, Mobile Fuzzy Logic Controllers

**Received:** 17 July 2013, Revised 24 August 2013, Accepted 30 August 2013



## 1. Introduction

Fuzzy logic expert systems have been used for variety of applications [1, 2]. This paper is on using FLES to solve control problems [3]. Within in these control problems this paper focuses on developing mobile fuzzy logic control systems FLCS [4, 5], as the world is moving rapidly-towards mobile-applications that require mobile control. For these mobile controllers fuzzy logic controllers FLCs are attractive because they are robust, multiple in and multiple out, and simpler to implement. FLCs can be implemented on PCs or on main-frame computers or on microcontrollers. To make them mobile one should use microcontrollers with battery back-up. If microcontrollers are used, the commercial FLES-software packages cannot be used as they cannot be installed on them.

One needs to write their own FLES-simulator-code to run on them. The software presented is in this paper is one such code to run on the microcontrollers. In this paper section 2 includes a review of some of the microcontroller development kits that are available today. Section 3 outlines FLES architecture for control applications. Section 4 describes the execution sequence of the inference engine. Section 5 includes test results and section 6 the conclusions. The following sections describe this FLES-simulator whose code is written in C++.

**2. Microcontroller Development KITs**

There are a large number of inexpensive microcontroller development kits MDKs available today. For selection of a kit one would consider two primary features other than its cost: 1. the processor or the controller or the microcontroller unit MCU; and 2. the integrated development environment IDE. The processor performs all the computing and IDE is used develop and download programs into the controller. IDE can also be used for off-line simulation of the system. The controller can be specified by its: a. number of bits-8-bit, 16-bit, 32-bit, 64-bit; b. architecture – advanced risk machine ARM, Harvard, advanced virtual risk AVR (modified Haward), complex instruction set computing CISC (power architecture). The IDE can be: KEIL – it is a common platform used by many controller kits, it supports C/C++; many of the other IDEs are development-kit specific. The ARM-company hold patent on the ARM-architecture it also owns patent on Keil-IDE. Many controller manufacturers and kit manufacturers have bought the patent-licensing from the ARM-company. ARM is the patent holder as well as the manufacturer of the RISC-processors. The best among the processors are 32-bit RISC; and among the IDEs Keil is desired with its C/C++ support, but it is not essential. Examples of AVR-processors are: Atmel – A, B-8-bit, C, D-32-bit series. The examples of the 32-bit RISC-processors are: ARM/Atmel – Cortex M0, M3, M4 series; Apple – A5, A6-32-bit, A7-64-bit. A7 is the most powerful processor available today and is used in iPone5s.

Some examples of frequently used MDKs include:

| MDK | MCU | IDE | Cost |
|---|---|---|---|
| Arduino Due | Atmel Cortex-M3 | Atmel Studio or AVR Studio | $50 |
| TI LaunchPad C2000: Picolo | ARM Cortex-M3 | CCSv5 + Control suite | $17 |
| STMicroelectronics F3-series | ARM Cortex-M4 | Keil + | $319 |

**3. Mobile FLEs For Control Applications**

As shown in Figure 1, the five basic elements of the FLES being: Knowledge base KB, Inference engine IE, User interface UI, Input-fuzzifier, and output-defuzzifier. The Knowledge base is a set of empirical rules by which the overall system behavior can be summarized. In an empirical rule the input/output values are in linguistic-form such as positive-small ps, negative-large nl, and positive-medium pm. Inference engine is the kernel of the FLC, that is, it executes all of the sequential steps involved in the overall execution of the FLC; starting from reading the inputs until the control outputs are generated. These steps are described in the following section. Input-fuzzifier converts absolute values into linguistic values. Here the input absolute values are expressed as a function of input membership functions IMFs. Output-defuzzifier converts fuzzy or linguistic values into absolute values. The computing system is the one on which the FLC-kernel or FLC-executive is executed. For off-line applications it is usually a personal computer. For on-line mobile applications, it should be a microcontroller.

**4. The Software Architecture**

Figure 2, shows the overall architecture of the embeddable software that is developed. It has three-input modules and one-output module. The first input module has sensor values in absolute form. The second input module has membership functions of input and output variables. The third input module has the knowledge base of the system in the form of rules. The output-module has the defuzzified output control signals to go to the process control devices PCDs.

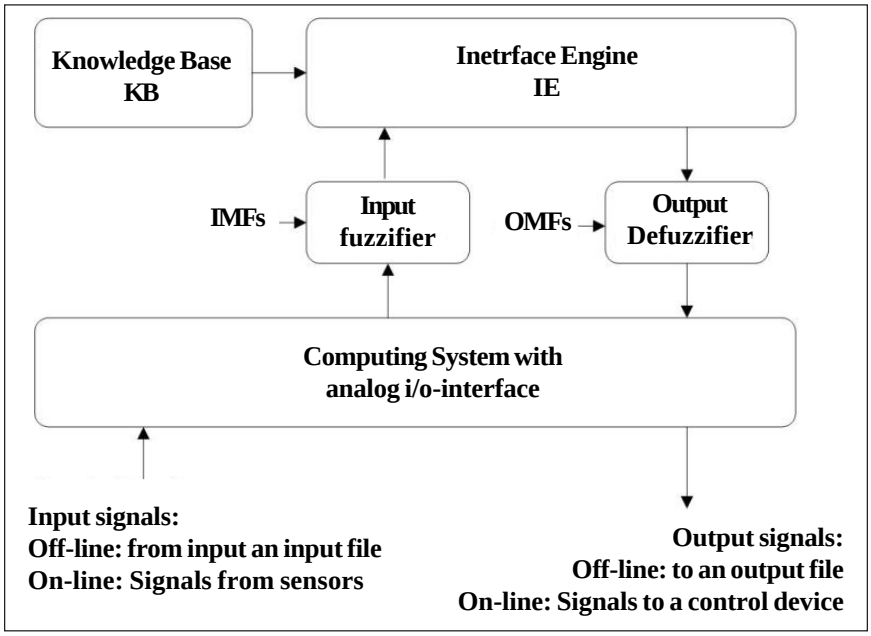


Figure 1. Basic Elements of an FLES for Control Applications

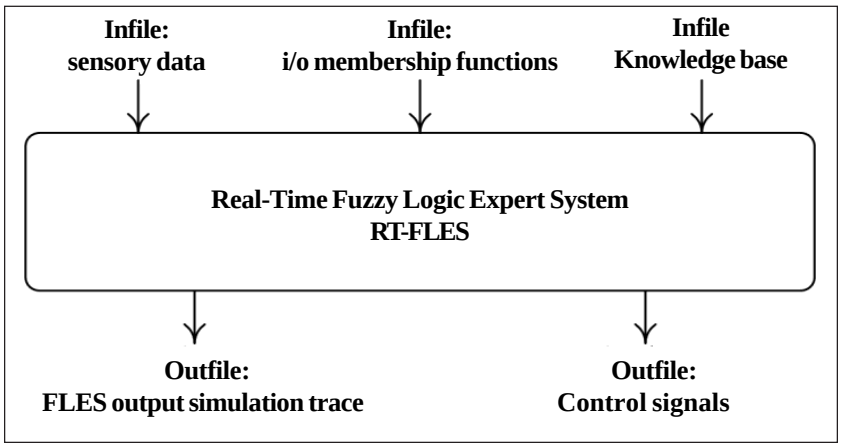


Figure 2. Embeddable FLES-Software Architecture

While running off-line (in simulation-mode) on a personal computer, the inputs are read from three input files and the output is stored into an output file. While running on-line on a micro-controller, the input knowledge base and the membership functions are merged into the C++ code. Sensory inputs are mapped to the analog-input-ports of the controller; and the outputs are mapped to a controller's analog output-ports. The overall execution sequence of the inference engine, to implement FLC, is detailed in the following section.

**5. FLC Kernel: Execution Sequence of the Inference Engine**

Step-by-step sequences of tasks executed by the inference engine are as follows:

*IE*1: *Fuzzify input variables*

Express input variables as a percentage of the input membership functions IMFs.
Translate inputs into linguistic form.

*IE*2: *Find the activated rule-set Ra*

Find the rule set *Ra* in the KB, where their input-variable-value requirements match with the current input-values in their

linguistic form.

*IE*3: *Find effective input membership function eimf-Ri for each of the activated rule-Rai.*

Example: assuming there are two input variables in1 and in2 and the rules are logically ANDed such as:

if (in1 is *ns*) AND (in2 is pm) then *y* is *nm.*

eimf-Rai = min (in1v-Rai, in2v-Rai),

eimf-Raj = min (in1v-Raj,in2v-Raj),

where eimf-Rai is the effective input membership function of the active rule Rai. in1v-Rai and in2v-Rai are the fuzzified values of in1 and in2 in Rai. Similarly, eimf-Raj is the effective input membership function of the rule Raj; in1v-Raj and in2v-Rj are fuzzified values of in1 and in2 in Raj. The fuzzified variable values in1v and in2v being expressed as a percentage of the input membership functions. To find eimf, select minimum of in1v and in2v.

*IE*4: *Find the final rule to execute with its eimf and eomf*

among the active-rules identify the rule to execute: Rule to fire *Rf = R* with: max (eimf-Rai, eimf-Raj,…)

a. Select the rule Rf corresponding to the maximum of the eimfs

b. eimf-Rf = min (in1v-Rf, in2v-Rf)

c. eomf-Rf = omf-Rf (with one output variable) with multiple output variables eomf should be computed similar to the eimf.

*IE*5: *Find defuzzified control output y*

$output = eimf * eomf$

$y = eimf\text{-}Rf * eomf\text{-}Rf$

The embeddable software presented in this paper executes these five steps in sequence. It is also generates an output simulation trace corresponding to each of these steps. This output trace is illustrated in the following section for solving a specific control problem.

**6. Test Results: The FLEs Output Simulation Trace**

Test results of the software are presented in the form of solving a specific control problem. The control problem of this case study is to balance the inverted pendulum IP using a cart driven by *dc*-motors as shown in Figure 3. The function of the controller is to keep the IP straight-up. When the pendulum tilts away from the center by $\theta$ (angle a), degrees, at a rate of $d\theta/dt$ (derivative of the angle *da*); then the controller must generate a control signal to move cart by *x*-units in the right direction at right rate of $dx/dt$. The movement of the cart is proportional to motor current *mc*. If the controller is controlled at fixed interval then $dx/dt$ is not needed. The problem now has two input variables *a* and *da* and one output variable *mc*. The FLES configuration of this problem is shown in Figure 4. The input membership functions IMFs for the input variables *a* and *da* are shown in Figure 5. The triangular MFs are represented by three vertices in the output simulation trace as shown in Table 4. For example, imf-nm has vertices *p*0, *p*1, *p*2: –54, –36, –18. The fuzzified or linguistic value of the variable is "*nm*". Input variables *a* and *da* have min/max values of: –54 / +54; and units of degrees and degrees/sec respectively. The range is from –54 to +54 with 7 membership functions each. Figure 6 shows the output membership function OMF for the output variable mc. It also has 7 membership functions but they are singleton. The knowledge base used for the IP-problem shown is shown in Figure 7. The final goal now is for a given sensor values of *a* and *da* the FLC must generate the appropriate motor current *mc*.

In the software: the sensed values of *a* and *da* are specified through an input file "*infile1-SenData.txt*" as shown in Table 1. The input/output membership functions are specified through the input file "*infile2-IOMFs.txt*", refer to Table 2. The knowledge base is specified through another input file "*infile3-KB.txt*", this is shown in Table 3. The generated output is stored into an output file "*fles-outfile.txt*", this file is shown in Table 4.

While operating in real-time the knowledge base and the MFs are merged into the fles-C++ code. Sensed input values and the generated outputs are mapped to i/o-ports of a microcontroller. Depending on the problem everything can be tailored accordingly: the name and units of input and output variables; IMFs and the number of IMFs, OMFs and the number OMFs, and finally the KB.

The detailed results of execution tasks IE1 through IE5 of the inference engine, specified in section IV, are shown in Table 4. In

this table, the initial sections A, B, and C will display input data to the FLC. This includes the sensor data of the input variables; the input/output membership functions; and the knowledge base of the problem. The task here is to build an FLC to balance an inverted pendulum problem. It is a two-input and one-output problem. The input variable values for the FLC are: the pendulum's angular displacement (angle a) is –12.0 degrees away from the center, and rate of angular displacement (derivative of angle da) is –3.0 degrees/sec. Input variables in1 and in2 both have 7 membership functions. Minimum and maximum values are –54 to +54 with 18-units between the vertices with units of degrees for in1 and degrees/sec for in2. The output variable *mc* has 7-MFs with minimum and maximum of –18 ma and +18 *ma* with 6 ma separation. The *mc* linguistic values ranging from negative-large (*nl*) to positive-large (*pl*). The knowledge base of the IP-problem is specified by 13-rules denoted as R0-to-R12. These are logically ANDed-rules.

The 5-step procedure by which the inference engine generates the control signal from the sensor input data is shown in section-D of Table 4. The five steps are denoted as: IE1 through IE5.

**The IE1-Trace:** In IE1 the input variables are fuzzified: for in1 = –12 degrees, the fuzzified values are:
(12/18) – *ns* or (6/18) – *zr*; and for in2 = –3 degrees/sec, the fuzzified values are: (3/18) – *ns* or (15/18) – *zr*.

**The IE2-trace:** In IE2 active rule set Ra is determined:
the activated rules are R7 and R8. Ra = {R7, R8}.

**The IE3-trace:** In IE3 effective input membership functions eIMF-Rai for each active rule Rai is determined:
eIMF-R7 = (12/18) – *ns*;
eIMF – R8 = (6/18) – *zr*;

**The IE4-trace:** In IE4 Rule to be fired is determined:
Rule fired = max (eIMFs) = eIMF-R7 = (12/18);
Rule fired is R7

**The IE5-trace:** In IE5 find output – motor current *mc*:
output = eIMF * eOMF = (12/18) * *ps*
= (12/18) * 6.0 = 4.0 mille-amps

**7. Conclusions**

This paper presents FLES-C++ code that can be downloaded into microcontrollers to build mobile-FLCs. We have used this software to implement two fuzzy logic systems, one is an estimation problems and the other a control problem. Application one is used for precise estimation of the state of charge SoC of a battery using the impedance interrogation method [5]. For this we have used "*Handy board* " – a MC6811 micro-controller board. It supports "*Interactive-C*", for that we have converted C++ to C and then C to Interactive C[5]. The other is to implement the classical controller for balancing of an inverted pendulum. This is run through a personal computer with NI-ADC/DAC-card, therefore C++ was used as is with I/Os mapped to NI-DAQ card. Current controller, such as PIC-controllers, cards are very powerful and inexpensive of the order of less than $100. These controller-boards drain very little current as well, therefore, one can have battery backup system that can function for hours. With your own C++ code and powerful and inexpensive today's microcontrollers, one can build extremely complex yet inexpensive mobile FLCs.

**References**

[1] Antonelli, G., Chiaverini, S., Fusco, G. (2007). A Fuzzy-Logic-Based Approach for Mobile Robot Path Tracking, *IEEE Transactions on Fuzzy Systems*, 15 (2) 211-221, April.

[2] Hamzah Ahmad etal. (2012). Adaptive Speed Control for Autonomous Mobile Robot Using Fuzzy Logic Controller, International Conference on Intelligent Robotics Automation and Manufacturing IRAM-2012, p 67-74.

[3] Shu-Hsieen Liao. (2005). Expert System methodologies and Applications – a decade review from 1995 to 2004, *Expert Systems with Applications*, 28 (1) 93-103, Elsevier.

[4] Yager, R. R., Lofti Zadeh. (1992). An Instrodcution to Fuzzy Logic applications on in Intellegent Systems, Kluwer Academis

[5] Chuen Lee. (1990). Fuzzy Logic in Control Systems: Fuzzy logic Controller – Part I, IEEE International Conference on Systems, Man, and Cybernetics, 20 (2) 404-418.

[6] Chuen Lee. (1990). Fuzzy Logic in Control Systems: Fuzzy logic Controller – Part Part II, IEEE International Conference on Systems, Man, and Cybernetics, 20 (2) 419-435.

[7] Maila, Sreelatha (G. N. Reddy). (2008). Embedded-System Controlled Fuzzy Logic Expert System to Estimate Battery-SOC, Summer II, Electrical Engineering, Lamar University.