Prace Naukowe Instytutu Maszyn, Napędów i Pomiarów Elektrycznych
Nr 69 Politechniki Wrocławskiej Nr 69

Studia i Materiały Nr 33 2013

nonlinear control, expert systems, reasoning, AI

Paweł PTASZNIK*

EXPERT SYSTEM APPROACH IN DESIGNING KNOWLADGE-BASED CONTROLLER

In this article the knowledge-based control approach will be proposed for nonlinear systems. Firstly some basic concepts from the artificial intelligence will be defined, then the expert system design process will be introduced and finally a practical example will be discussed. The principle of this method is to support the controller with the human-like reasoning to determine the appropriate control mode and to estimate unknown parameters. Note that this is the heuristic approach, hence neither optimal steering, nor stability is guaranteed.

1. INTRODUCTION

It is relatively frequent situation for control system designers to realize, that the object which seemed to be easily driven by a human is extremely difficult to stabilize automatically. The mathematical model of such an object is usually either not complete or it depends on large amount of unknown parameters, e.g. mass, moment of inertia, resilience. Model equations may be so complex, that not only will the stability analysis be impossible, but also designed controller will not reproduce the given state values. Nonetheless, human being is still able to control the considered structure, because of its control methods, that are not based on mathematical dependencies, but on knowledge about the object. Some may wish to call it *intelligence*. According to [1], there is a significant distinction between data and knowledge. Modern computers are able to process data at very fast rates, but the processed information must be appropriately structured. This is a serious limitation, because our knowledge about systems we use is mostly unstructured. For example the Internet is almost inexhaustible source of data, but the machines are hardly able to obtain any significant

* Wrocław University of Technology, Institute of Electrical Machines, Drives and Measurements, ul. Smoluchowskiego 19, 50-372 Wrocław, e-mail: pawel.ptasznik@pwr.wroc.pl

information using it. It might be said, that human brain works in just the opposite way, i.e., it is rather slow in performing computations, but introduces incredible performance in searching for the information in almost infinite size database. Half-joking it might be said, that human brain has the O(1) search algorithm implemented. In the next part of this scripture the Knowledge-Based Expert System (KBES) will be presented as a tool that compromises between data structuring and human-like performance of a controller.

2. KNOWLEDGE-BASED EXPERT SYSTEM

After [4] Knowledge-Based Expert System is the most popular and successful of the artificial intelligence based tools that have been evolved to address problems in planning, diagnosis, classification, monitoring and design. In Expert Systems a few types of knowledge representation is available, but for clarity in this article the *production rules* and *facts*, sometimes called *triplets*, will be used. Inference mechanisms also vary and are extensively described in [2], [4] and [3]. The software tool for building expert systems described in this paper uses *forward chaining*.

The database, i.e., the part of the system responsible for data acquisition is usually separated from main reasoning engine. It is a good practice, because it permits to modify one part without violating the structure of the latter. The knowledge acquisition can be defined as a transfer and transformation of expert knowledge, which can potentially solve the desired problem, from some source to the system. This particular problem refers to the subfield of AI called *machine learning* and machines that learn vary in terms of information obtaining. The knowledge might be:

1. introduced manually by a programmer,
2. obtained by data manipulation,
3. obtained from the set of actual data by induction,
4. obtained via empirical discovery of new rules.

There are multiple historical examples [2] of expert systems using each of those methods, and might one be interested in studying them thoroughly, he should be able to easily find the required information in the literature. The nonlinear control system proposed in this elaboration should consider using of the first and the second method mentioned above.

One substantial property of expert systems, that distinguish them from other artificial intelligence structures, like neural networks, is the possibility of backtracking the reasoning chain. This means, that the control system designer is always able to inspect the path, which leaded the reasoning system to the particular conclusion. In other words, the control system debugging and adjusting is relatively easy. In the next section the CLIPS environment for developing expert systems will be introduced, as well as some basic examples will be provided.

3. BUILDING EXPERT SYSTEM WITH CLIPS

CLIPS (C Language Integrated Production System) is a public domain software tool for building expert systems. It was originally designed in 80s of the twentieth century by NASA scientists, but the latest version CLIPSJNI v0.3 written in Java is dated on 04.03.2011. There also exists an extension supporting *fuzzy logic* called FuzzyCLIPS.

The CLIPS database is the set of facts, that are declared with *assert* function. The *facts* function is used for inspecting the database content.

```
CLIPS> (assert (VelocitySensor1 15))
<Fact-1>
CLIPS> (assert (VelocitySensor2 0))
<Fact-2>
CLIPS> (facts)
f-0 (initial-fact)
f-1 (VelocitySensor1 15)
f-2 (VelocitySensor2 0)
For a total of 3 facts.
```

As already mentioned, the reasoning is performed by *forward chaining* method. The CLIPS engine uses the existing facts to apply* the appropriate rule. The rules of the form

```
IF conditions are true
THEN execute the following actions
```

are exactly the expert knowledge, which should be provided by the control system developer. The more precise rules are provided to the program, the greater potential efficiency of the system should be noticed. On the contrary, the facts should be provided rather by the controlled object, then by a person. Only one rule may be fired at a time, consequently should there be some particular rules, that conflict with each other, the embedded conflict resolution algorithm will be used. For more information about conflict resolving the reader is referenced to [5] and [3].

To illustrate the reasoning mechanism of the CLIPS inference system one may wish to imagine the small part of the nonlinear control system, which is responsible for velocity sensor failure detection. For simplicity it is assumed, that the difference

* In CLIPS nomenclature the term *fire* is used for applying the appropriate rule.

between two sensors indications implies the crash of one of them. In this particular part of the system the two rules should be defined:

```
CLIPS> (defrule MAIN::SensorMatch
(VelocitySensor1 ?s1)
(VelocitySensor2 ?s2)
(test (>= (abs (- ?s1 ?s2)) 0.001))
=>
(assert (VelocitySensor crashed)))
CLIPS> (defrule MAIN::EmergencyControl
(VelocitySensor crashed)
=>
(assert (use-controller emergencycontroller)))
```

Those rules may be introduced to the CLIPS using the following syntax

IF Sensor indications differ	IF Sensor crashed
Then sensor crashed	then use emergency control

Having the facts about velocity sensors indications defined, the program may be executed and the following output should be produced

According to the facts 1 and 2, rule 1 (SensorMatch) is fired. This rule produced new entry indicating the velocity sensor crash and therefore rule 2 (EmergencyControl) is applied. Note that despite the fact that facts 1 and 2 are still present at the end of the program execution, the rule 1 is not fired again. This is due to internal conflict resolution system, that prevents from firing the rule twice on the same set of data. It is also worth noticing, that CLIPS may provide the user with the complete data manipulation history (backtracking), which was mentioned in the previous section.

4. CLIPS INTEGRATION TO THE CONTROL SYSTEM

In the previous section the basic abilities of the CLIPS inference have been shown. The simple examples have been handled using CLIPS Java command line interface, where the definitions had to be introduced manually to the system. Obviously, this way is comfortable, when one wish to demonstrate some basic activities, but for the control system design purposes the reasoning module must be integrated with the remaining part of the project.

According to [6], there is a possibility of building a CLIPS native library with the C-language sources provided within a CLIPS package. This method may be efficient enough, assuming that the control system will be used on the same platform, on which

it is already in development. If not, it is better to consider integrating the CLIPS utilities with the control system so that it will be platform independent. In this paper the CMake build system is proposed for this purpose.

As shown in the Figure 1, the Expert System should support the Control System, as a separate unit, with decision making process. Therefore it is a good practice to separate its sources from the main project tree, as shown in the Figure 3. Files *CLIPSJNI_Environment.h* and *CLIPSJNI_Environment.c* should not be taken into account, because they are used for building Java interface executable, which is naturally not desired in this case. The corresponding CMake project files should have the statements listed in the Figures 2a and 2b included. The *desired flags* may be another flags facilitating the development. In the original CLIPS makefile, for example *-Wall* is proposed along with some other with similar meaning.

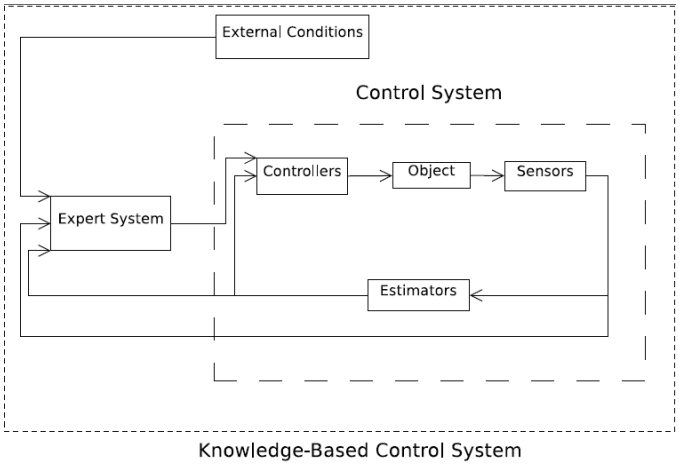

Fig. 1. Graphical representation of Knowledge-Based Expert System

CMake is an extensible, open-source system that manages the build process in an operating system and in a compiler-independent manner [7], consequently it is suitable for control purposes, especially when designed algorithms need to be implemented in the object.

With such configured build system it is safe to develop the controller without worrying about its portability. However, during the development process one should take deeply into consideration remarks about memory management in CLIPS program [6]. It is extremely unsafe to store the pointers returned from CLIPS in the local memory, because in the inference system the garbage collector[†] mechanism is implemented.

[†] Garbage collection is a form of automatic memory management. The collector attempts to reclaim memory occupied by objects that are no longer in use by the program.

```
make_minimum_required( VERSION 2.8 )
project(CLIPS)
include_directories(include include/libCLIPS)
add_subdirectory(src)
```

(a) ./CMakeLists.txt

```
file(GLOB LIB_CLIPS_SRCS
${CMAKE_CURRENT_SOURCE_DIR}/libCLIPS/*.c)
set(CMAKE_CXX_FLAGS "-std=c99 -another_desired_flags")
set(CMAKE_EXE_LINKER_FLAGS "-lm")
add_library(libraryCLIPS STATIC ${LIB_CLIPS_SRCS})
add_executable(main main.c)
target_link_libraries(main libraryCLIPS)
```

(b) ./src/CMakeLists.txt

Fig. 2. CMakeLists files for sample project

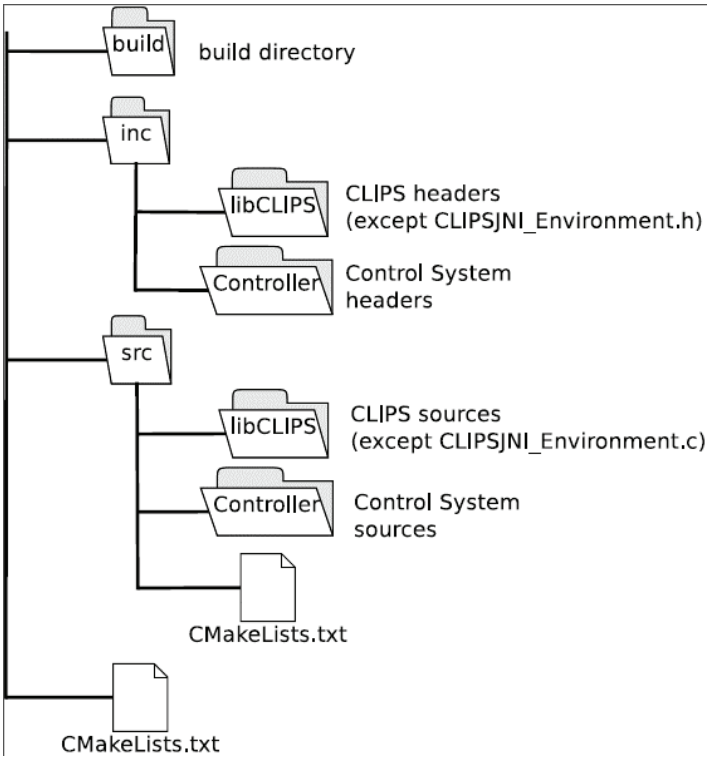

Fig. 3. Proposed directory structure

5. PRACTICAL EXAMPLE

So far the basics of the expert systems design have been presented as well as some system integration steps have been proposed. A rule example partially indicates the potential application field, however some may still be wandering, what are this expert systems really useful for. Nevertheless, one may implement an expert system-like behavior using *if-then* statements, whatever programming language he uses. Of course it is true, but not with extremely huge knowledge base. With big amount of rules to process *if-then* statements become difficult to maintain, and then the expert system should be used.

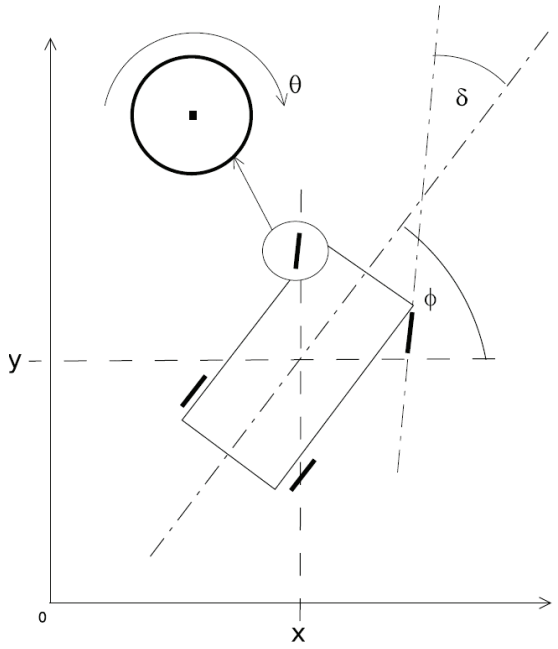

Fig. 4. Graphical interpretation of vehicle states (1)

As an example the experimental autonomous vehicle control system will be briefly introduced. The vehicle is considered as an object with 5 states distinguished centered.

$$q = (x, y, \phi, \delta, \theta)^T . \tag{1}$$

The graphical interpretation of the coordinates (1) is presented in the figure 4. X and Y are the Cartesian coordinates, Φ is the orientation of the car, δ is the orientation of the front wheels with respect to the body, and θ is a turn angle of the

front wheels. State δ may also be identified with steering wheel turn angle, and θ may be identified with the tire valve angular position. Motion equations of such an object may be obtained by defining the non-holonomic constraints [8]:

$$\begin{gathered}
\dot{x}_p \sin(\Theta+\delta) - \dot{y}_p \cos(\Theta+\delta) = 0, \\
\dot{x}_t \sin(\Theta) - \dot{y}_t \cos(\Theta) = 0, \\
\dot{x}_p \sin(\Theta+\delta) - \dot{y}_p \cos(\Theta+\delta) - R\dot{\phi} = 0 \\
\text{where} \\
xp = x + L\cos\Theta \qquad yp = y + L\sin\Theta \\
xt = x - L\cos\Theta \qquad yt = y - L\sin\Theta \\
L - \text{half of the vehicle length} \quad R - \text{wheel radius}
\end{gathered} \tag{2}$$

That constraints may be presented in the matrix form also known as Pfaff form.

$$A\dot{q} = \begin{bmatrix} \sin(\Theta+\delta) & -\cos(\Theta+\delta) & -L\cos\delta & 0 & 0 \\ \sin\Theta & -\cos\Theta & L & 0 & 0 \\ \cos(\Theta+\delta) & \sin(\Theta+\delta) & L\sin\delta & 0 & R \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\Theta} \\ \dot{\delta} \\ \dot{\phi} \end{bmatrix} = 0. \tag{3}$$

Now, by calculating the null space G of matrix A the complete motion equations of the vehicle are obtained.

$$\dot{q} = Gu = \begin{bmatrix} -\frac{1}{2}R(\cos\delta\cos\Theta + \cos(\Theta+\delta)) & 0 \\ -\frac{1}{2}R(\cos\Theta\sin\delta + 2\cos\delta\sin\Theta) & 0 \\ -\frac{R\sin\delta}{2L} & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{4}$$

As one can see, the first control input u_1 may be interpreted as an angular velocity of front wheels, and the second control input u_2 may be interpreted as angular velocity of front wheels turning. Actually, in the First Polish Autonomous Vehicle experimental Knowledge-Based Control System not only were the kinematic equations considered, but also the dynamics has been calculated using Euler–Lagrange formalism. However, it is not necessary to introduce those complicated dependencies in order to explain the idea of the reasoning part of the control system. In the figure 5 the graphical representation of autonomous vehicle control system is shown.

It is worth noticing, that even the most sophisticated trajectory tracking algorithms are not sufficient to consider such a car self-driving. In this particular case multiple controllers (not necessarily linear) are used and their main responsibility is to ensure that car is reproducing the referenced trajectory. Then there are vision system and radars, that are responsible for detecting the traffic conditions and gathering useful information, e.g. road signs. That will be extremely difficult to produce multiple *if-then* rules in order to design adequate nonlinear controller, taking all the road conditions into consideration, therefore the expert system is used. The expert system rule base used in autonomous vehicle control system is not yet fully finished, but in that development stage one could divide them into several categories, which are:

- Human inquiry-based: That category consists of rules, that process the reference values given by human, such as: destination point, preferred road options and some velocity constraints. For example, given destination point is forwarded to the path-resolving algorithm (A* in most cases) and then, considering the velocity constraints and other events, reference trajectory is calculated.
- Radar-based: This category consists of rules, that are triggered by facts asserted by radar module. The sample facts may be: another vehicle detected, lane blocked, or distance to the obstacle. The reasoning system may react with asserting velocity constraints, recalculating the path or even emergency stopping the car.
- Sign-based: This category of rules is strongly dependent on visual system, that is responsible for sign detecting. Whenever the fact indicating the new road sign is asserted, expert system reacts with deploying appropriate constraints, such as speed limit, recalculating the path (e.g., when left turn is forbidden), or just communicating with human (e.g., petrol station in 20 km).
- System-based: This category consists of rules, that should fire, whenever some system event id detected. The system event may by understood as any control system dependent anomaly, such as extremely different sensor indications, sensor providing an error code, or persistent trajectory tracking error. The desired actions are, e.g., trajectory tracking algorithm alternation.

The sample rule responsible for applying the velocity constraint after detecting the speed-limit sign is presented below.

This rule searches the database for appropriate facts. The first should represent the vehicle moving with some velocity, present on some road lane, performing some action and having lights on or off. The latter should represent the speed limit sign. When the rule fires, it erases the fact representing the vehicle, and replaces it with the new one, with new velocity asserted. It also provides the human with warning message. In the end it erases the fact associated with speed limit sign to prevent its future unwanted usage.

```
(defrule sign-speedlimit
?v <- (volvo (velocity ?)(lane ?ln)(action ?act)(lights ?lgt))
?z <- (sign velocity ?limit)
=>
(retract ?v)
(assert (volvo (velocity ?limit)(lane ?ln)(action ?act)(lights ?lgt)))
(assert (statement velocity (str-cat "ATTENTION: Speed limit: "
?limit)))
(retract ?z)
)
```

5. SUMMARY

This article hardly covers the most basic expert systems principles and do not respond to the question of how to fully design the knowledge-based nonlinear controller, but naturally that was not the objective. This elaboration should encourage the engineers, who have faced the problem with control system design, to use the working knowledge-based solutions, rather than defining whole reasoning system on their own. That should accelerate the development process. For those who wish to implement their ideas to the CLIPS, positions [2] and [3] are recommended.

The examples adduced in this work are very simple, so as they explain the main idea of the CLIPS reasoning engine. Both the knowledge base and the internal structures of the presented subsystems are very complicated, therefore it is not possible to describe them exhaustively. To implement good rules to the control system, one should consult his ideas with people experienced in the use of the particular object. Actually the decision-making system in described autonomous vehicle is developed by many engineers and it still needs to be adjusted.

REFERENCES

[1] *Clips reference manual*, Quicksilver Beta, March 22nd 2008.

[2] GIARRATANO J.C., *Clips user's guide*, 31st December 2007.

[3] GIARRATANO J.C., RILEY G.D., *Expert Systems: Principles and Programming Course Technology*, 1998.

[4] MARTIN K., HOFFMAN B., *Mastering CMake*, Kitware, Inc., September 2013.

[5] TCHOŃ K., MAZUR A., DULĘBA I., HOSSA R., MUSZYŃSKI R., *Manipulatory i roboty mobilne*, Akademicka Oficyna Wydawnicza PLJ, Warszawa, 2000.

[6] JACKSON P., *Introduction to expert systems*, International Computer Science Series. Wokingham: Addison-Wesley Publ. Co., 1986.

[7] SCHALKOFF R., *Intelligent Systems: Principles, Paradigms and Pragmatics: Principles, Paradigms and Pragmatics*, Jones & Bartlett Learning, 2011.