Embedding Fuzzy Controllers in Golog

Alexander Ferrein*†
*Robotics and Agents Research Lab
University of Cape Town, South Africa
Email: alexander.ferrein@uct.ac.za

Stefan Schiffer† and Gerhard Lakemeyer†
†Knowledge-Based Systems Group
RWTH Aachen University, Germany
Email: {schiffer, gerhard}@cs.rwth-aachen.de

***Abstract*— High-level behaviour specification of an intelligent autonomous agent or robot is a non-trivial task. Various approaches exist some of which try to combine different paradigms like programming and planning. In this paper, we show how to integrate fuzzy logic controllers into the logic-based programming language Golog. Golog already allows for combining programming and planning. By adding the instrument of fuzzy controllers we provide the means to have a natural specification of rules for tasks that require a high amount of reactivity. Since the facilities already present in Golog remain, we add to an already powerful framework thus expanding the applicability of Golog for high-level behaviour specification of a robot or agent.**

I. Introduction

In the last decades there have been several approaches on how to specify the high-level behaviour of an agent or robot. Especially in domains where a fast reaction time is required approaches such as planning tend to fail due to their increased computational complexity. That is why one often seeks to combine reactive control with high-level deliberation trying to get the best of both worlds. Previously, Golog [7] was used for controlling robots acting in the real world. The paradigm was to combine deliberation with explicit programming. Now, we combine Golog with fuzzy control as a means for reactive decision making while not losing the advantages of deliberation and a logic-based robot controller framework.

Fuzzy logic was deployed successfully for several robotics tasks. As an example for the large body of work dealing with fuzzy control and robots, we want to mention [14] where Soffiotti shows how fuzzy controllers can be used to design robust behaviour-producing modules, and even how high-level reasoning and low-level execution can be integrated on a mobile robot. He attributes the success of fuzzy logic in control to "its ability to represent both the symbolical and the numerical aspects of reasoning. Fuzzy logic can be embedded in a full logical formalism, endowed with a symbolic reasoning mechanism; but it is also capable of representing and processing numerical data." Liu et al. [9] describe a robot kinematics in a qualitative fashion making use of fuzzy descriptions. They use fuzzy qualitative trigonometric functions to describe the movements of a PUMA robot manipulator. According to the authors this fuzzy qualitative description is very helpful for calibration procedures in terms of measuring accuracy or repeatability. They further stress that the fuzzy qualitative predicates provide the connection between the numerical data and interval symbols, which are then used for building up the behaviour vocabulary from which in turn the motion control of the robot can be derived. In [8], Liu uses the forementioned fuzzy kinematics to close the representational gap between the quantitative methods applied to drive the PUMA arm, and a logical representation based on fuzzy logic predicates. Many other successful examples for fuzzy control applications are given in [6]. Good overviews of the fields are also given in [3], [10], [11], [15].

The usefulness of fuzzy sets to make use of qualitative fluents in the situation calculus was already shown in [5]. Now we take a step further to not only give qualitative descriptions by means of a fuzzy set semantics, but based on this define fuzzy controllers in Golog.

II. Fuzzy Sets and the Situation Calculus

A. The Situation Calculus

The situation calculus is a second order language with equality which allows for reasoning about actions and their effects. The world evolves from an initial situation due to primitive actions. Possible world histories are represented by sequences of actions. The situation calculus distinguishes three different sorts: *actions*, *situations*, and domain *objects*. A special binary function symbol $do : action \times situation \rightarrow situation$ exists, with $do(a, s)$ denoting the situation which arises after performing action a in the situation s. The constant S_0 denotes the initial situation, i.e. the situation where no actions have occurred yet. The state the world is in is characterised by functions and relations with a situation as their last argument. They are called *functional* and *relational fluents*, resp. The third sort of the situation calculus is the sort *action*. Actions are characterised by unique names. For each action one has to specify a *precondition axiom* stating under which conditions it is possible to perform the respective action and an *effect axiom* formulating how the action changes the world in terms of the specified fluents. For space reasons we will not go into the details on the formalisation of the situation calculus here. To get an idea, it is enough to know that actions can only be performed when they are possible, i.e. their precondition axiom holds, and their effects are manifested in the environment. Finally, we need a basic action theory, which is a set of sentences $\mathcal{D}$ consisting of $\mathcal{D} = \Sigma \cup \mathcal{D}_{ssa} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, where $\mathcal{D}_{ssa}$ contains sentences about the successor state axioms, $\mathcal{D}_{ap}$ contains the action precondition axioms, $\mathcal{D}_{una}$ states sentences about unique names for actions, and $\mathcal{D}_{S_0}$ consists of axioms what holds in the initial situation.

Additionally, Σ contains a number of foundational axioms defining situations. For details we refer to [12], [13].

B. Golog

The high-level programming language Golog [7] is based on the situation calculus. As planning is known to be computationally very demanding in general, which makes it impractical for deriving complex behaviours with hundreds of actions, Golog finds a compromise between planning and programming. The robot or agent is equipped with a basic action theory. The programmer can specify the behaviour just like in ordinary imperative programming languages but also has the possibility to project actions into the future. The amount of planning (projection) used is in the hand of the programmer. With this, one has a powerful language for specifying the behaviours of a cognitive robot or agent.

As an example consider a robot with the goal to deliver a letter to a user Alex. A Golog program for this tasks could be:

proc deliver_letter
  navigate($post\_office$); pick_up($letter$);
  if $inOffice(alex)$ **then** deliver_letter($alex$)
  else do_nothing **endif**
endproc

This simplistic program shows some sequences of primitive actions as well as control constructs like if-then-else or procedures. Such programs are evaluated using a transition semantics, making use of the situation calculus. This is intriguing as we have a formal situation calculus semantics for our programming and planning language and the execution trace yields a formal proof of the executability of the program. With such a basic action theory tasks like projecting programs into some future world situation becomes possible.

The program is interpreted in a step-by-step fashion where a transition relation defines the transformation from one step to another. In this so-called transition semantics a program is interpreted from one configuration $\langle \sigma, s\rangle$, a program σ in a situation s, to another configuration $\langle \delta, s'\rangle$ which results after executing the first action of σ, where δ is the remaining program and s' the situation resulting from the execution of the first action of σ. The one-step transition function $Trans$ defines the successor configuration for each program construct. In addition, another predicate $Final$ is needed to characterise final configurations, which are those where a program is allowed to legally terminate. To give an example we show the definition of the conditional:

$$Trans(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s, \delta, s') \equiv$$
$$\phi[s] \wedge Trans(\delta_1, s, \delta, s') \vee \neg\phi[s] \wedge Trans(\delta_2, s, \delta, s')$$

We only sketched the transition semantics here. For a concise overview of the transition semantics we refer the interested reader for example to [1], [2]. We remark that the transition semantics allows for a natural integration of sensing and on-line execution of programs. Further, we want to note that our implementation is based on another variant of Golog: Readylog. In particular, we need on-line execution and exogenous fluent features from that dialect, though it suffices for the moment to know that Readylog is a Golog dialect with a transition semantics which allows a direct coupling of sensor values to the run-time system. As there is not enough space to get into the details here, for an overview of other features of Readylog, we refer to [4].

C. Fuzzy Fluents Revisited

We need to define qualitative categories, fuzzifiers and defuzzifiers, as well as possibilities to query and access qualitative/fuzzy values in the situation calculus. In [5], we defined a fuzzy set semantics for qualitative fluents in the situation calculus. In the following, we sketch the ideas from [5], as they are some of the key ingredients for defining the semantics of fuzzy controllers in Golog.

First, [5] introduces reals and linguistic terms in the situation calculus. Reals are to be understood with their standard calculus interpretations together with their usual operations, while linguistic terms are defined as constants in the situation calculus. Each constant refers to a unique qualitative class, that is $c \neq d$ for distinct c, d. A fuzzy set $\mathfrak{F} \subseteq linguistic \times real \times [0,1]$ is defined as

$$\begin{aligned}\forall c, u, \mu_u.\mathfrak{F}(c, u, \mu_u) \equiv\\ (c = c_1 \supset u = u_{c_1,0} \wedge \mu_u = \mu_{c_1,0} \vee \cdots \vee\\ u = u_{c_1,m_1} \wedge \mu_u = \mu_{c_1,m_1}) \vee \ldots \vee\\ (c = c_k \supset u = u_{c_k,0} \wedge \mu_u = \mu_{c_k,0} \vee \cdots \vee\\ u = u_{c_k,m_k} \wedge \mu_u = \mu_{c_k,m_k}),\end{aligned}$$

assigning each quantitative value belonging to a category c a non-negative membership value. One has to ensure that, for each category, each pair (u, μ_u) is unique, that is $\forall c, u, \mu_u, \mu'_u.\mathfrak{F}(c, u, \mu_u) \wedge \mathfrak{F}(c, u, \mu'_u) \supset \mu_u = \mu'_u$. In the following definitions, we assume the following t-norm, s-norm, and negation:

$$A \cup B \supset \mu_{A\cup B}(x) = \max[\mu_A(x), \mu_B(x)] \quad (1)$$
$$A \cap B \supset \mu_{A\cap B}(x) = \min[\mu_A(x), \mu_B(x)] \quad (2)$$
$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \quad (3)$$

Fuzzy fluents are defined as functional fluents which take values from $\mathbb{R}$. As the definition of membership is important here, we restate the definition from [5]:

Definition 1 (Membership) *To query if a fuzzy fluent* $\mathfrak{f}(\vec{t}, \sigma)$ *belongs to a given category* γ*, we define the predicate* $\text{is} \subseteq real \times linguistic$ *as*

$$\text{is}(\mathfrak{f}(\vec{t}, \sigma), \gamma) \stackrel{def}{=} \exists u, \mu_u.\mathfrak{f}(\vec{t}, \sigma) = u \wedge \mathfrak{F}(\gamma, u, \mu_u) \wedge \mu_u > 0$$

Similarly, we define $is_{\mathfrak{C}} \subseteq real \times linguistic$ *to mean that a fuzzy fluent does* not *belong to a certain category*

$$\begin{aligned}\text{is}_{\mathfrak{C}}(\mathfrak{f}(\vec{t}, \sigma), \gamma) \stackrel{def}{=} \neg\exists u, \mu_u.\mathfrak{f}(\vec{t}, \sigma) = u \wedge \mathfrak{F}(\gamma, u, \mu_u) \vee\\ \exists u, \mu_u.\mathfrak{f}(\vec{t}, \sigma) = u \wedge \mathfrak{F}(\gamma, u, \mu_u) \wedge \mu_u < 1.\end{aligned}$$

It holds that a fluent value does not belong to a certain category, if either the value in question is not defined in terms of a fuzzy set, or the value exists and its degree of membership is less than 1 (cf. also Eq. 3 for the definition of complement). For complex queries, for example if a fuzzy fluent value belongs to several overlapping categories at the same time, we define a predicate $\text{is}_\star \subseteq \mathit{real} \times (\mathit{linguistic})^n$ *with a finite* $n \in \mathbb{N}$ *as*

$$\begin{aligned}\text{is}_\star(\mathfrak{f}(\vec{t},\sigma),\gamma_0,\ldots,\gamma_n) &\stackrel{def}{=}\\ &\exists u,\mu_{u,0},\ldots,\mu_{u,n}.\mathfrak{f}(\vec{t},\sigma)=u\wedge\mathfrak{F}(\gamma_0,u,\mu_{u,0})\\ &\wedge\cdots\wedge\mathfrak{F}(\gamma_n,u,\mu_{u,n})\wedge(\mu_{u,0}\star\cdots\star\mu_{u,n}>0).\end{aligned}$$

Here, the t-norm $\star$ *refers to the min-operation as given in Eq. 2. Similarly, for asking whether or not a fuzzy fluent value belongs to one category or the other, we introduce the predicate* $is_\oplus : \mathit{real} \times (\mathit{linguistic})^n$ *with a finite* $n \in \mathbb{N}$

$$\begin{aligned}\text{is}_\oplus(\mathfrak{f}(\vec{t},\sigma),\gamma_0,\ldots,\gamma_n) &\stackrel{def}{=}\\ &\exists u,\mu_{u,0},\ldots,\mu_{u,n}.\mathfrak{f}(\vec{t},\sigma)=u\wedge\mathfrak{F}(\gamma_0,u,\mu_{u,0})\\ &\wedge\cdots\wedge\mathfrak{F}(\gamma_n,u,\mu_{u,n})\wedge(\mu_{u,0}\oplus\cdots\oplus\mu_{u,n}>0).\end{aligned}$$

In our case, the s-norm $\oplus$ *refers to the max-operator as given in Eq. 1.*

We now have everything we need to define fuzzy controller support in the Golog language. We define the semantics for fuzzy controllers in the next section.

III. Interpreting Fuzzy Rules in Golog

In this section we define the support for fuzzy controller in the language Golog. We make use of our dialect Readylog [4], as it comes with many useful features like on-line fluents or continuous change, and our implementation is based on Readylog. However, the formal semantics can be transferred to any other Golog dialect easily. Note that we use Golog as a synonym for Readylog. Before we start with defining the predicates, which map the common fuzzy inference rules into logic, we introduce new program statements for embedding fuzzy controllers in our control language and define their transition semantics formally.

A. Fuzzy Controller in Golog

In this section we show the embedding of a fuzzy controller in the Golog language. The general architecture of a fuzzy controller is shown in Fig. 1. The quantitative sensor values $y(t)$, together with some reference input $r(t)$, which describes the vital state of the system, need to be fuzzified, i.e. the membership to a certain linguistic class needs to be determined. The *Inference Mechanism* uses these fuzzified input values together with a rule base of fuzzy rules to select the appropriate control output. The output as such uses fuzzy categories and thus must be defuzzified to serve as an input $u(t)$ for the real world. The output of the real world process serves as the sensor input for the next control step.

Following Fig. 1, we need some means to do the fuzzification, apply some inference mechanism to our rule base and defuzzify the output again. Therefore, we introduce a new statement

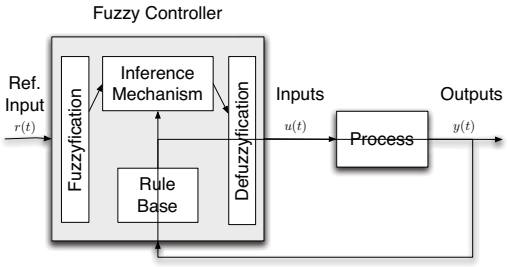

Fig. 1. General architecture of a fuzzy controller from [11]

$$\textbf{fuzzy\_controller}(\text{rule-base})$$

into our language. The rule base basically consists of conditionals in form of if-then statements which encode the fuzzy rules. These conditionals can be grouped in sequence. Moreover, we define default actions which will be selected as output in case no other rule matches. Thus, a rule base can be seen as a restricted Golog program consisting of a sequence of if-then statements and a number of default rules. For example, a rule base might look like

$$\begin{aligned}&\textbf{if } \phi \textbf{ then } assign(\mathfrak{f},c_k);\cdots;\\ &\textbf{if } \psi \textbf{ then } assign(\mathfrak{g},c_l);\\ &\textbf{default}(assign(\mathfrak{f},c_n);assign(\mathfrak{g},c_m))\end{aligned}$$

Formally, we define

$$\begin{aligned}&Trans(\textbf{fuzzy\_controller}(p),s,\delta,s')\equiv\\ &\quad\exists p',d.infer(p,p',d,s)\wedge s'=s\wedge\\ &\quad(Final(p')\wedge\delta=d\vee\neg Final(p')\wedge\delta=p')\end{aligned}\tag{4}$$

Note that free variables in formulae are implicitly universally quantified. The predicate $infer(p,p',d,s)$ is the interface to our fuzzy inference engine. The idea is that a certain fuzzy rule base p and some particular world situation s is handed over to the inference engine, and the result will be a sequence of output assignments for the control variables. In this way, we transform the rule base to one or more matching fuzzy outputs. The variable d defines some default output assignment, as we will describe below. We will defer the explanation of the $Final$ predicate in the definition above until we introduced the definition of conditionals and the default statement. For now it is sufficient to know that the remaining program δ will consist of one or more assignment actions which will assign a value to our control output variables.

To interpret a fuzzy conditional, we use

$$\begin{aligned}&infer(\textbf{if } \phi \textbf{ then } assign(\mathfrak{f},c),p',d,s)\stackrel{def}{=}\\ &\quad\phi[s]\wedge p'=assign(\mathfrak{f},c)\vee\neg\phi[s]\wedge p'=nil\end{aligned}$$

where $\phi[s]$ stands for the formula ϕ with the situation argument restored. If the antecedent ϕ of a fuzzy rule holds, then we will add the consequence $assign(\mathfrak{f},c)$ to our control output

p', otherwise the rule will contribute nil. Before we discuss how ϕ is evaluated, we need to define how sequences are defined in terms of the predicate $infer$. As stated above, our rule base consists of sequences of conditionals and defaults. Hence, we need to interpret sequences:

$$\begin{aligned} &infer((p_1;p_2),p',d,s) \stackrel{def}{=} \\ &\quad \exists p_1', p_2'.infer(p_1,p_1',d,s) \wedge \\ &\quad infer(p_2,p_2',d,s) \wedge p' = p_1';p_2'. \end{aligned}$$

Each matching fuzzy rule will be replaced by its consequence, i.e. the assignment statement, while non-matching ones contribute nil. Sometimes, it may happen that no given rule in a controller block matches at all, nevertheless some output would be required. We therefore define an additional statement $\mathbf{default}(assign(\mathfrak{f},c);\ldots)$, which is interpreted in case the control output was the nil action after evaluating the rule base. We allow one or more assignment actions as parameter of the default, which is denoted by the ellipsis:

$$\begin{aligned} &infer(\mathbf{default}(assign(\mathfrak{f},c);\ldots),p',d,s) \stackrel{def}{=} \\ &\quad d = assign(\mathfrak{f},c);\ldots \end{aligned}$$

With the predicate $infer$ we are able to interpret the statements which are allowed inside the rule base. The idea is pretty much the same as using Golog's evaluation semantics to find a legal action sequence (cf. e.g. [13]). Now it is clear why we used the $Final$ predicate in the definition of our fuzzy controller statement (Eq. 4). It holds $Final(nil;\ldots;nil,s) \equiv true$, i.e. if all rules in the rule base contributed nil, the final predicate becomes true and we choose a default output. If one rule matched, then the program p' in Eq. 4 contains an assignment action for which $Final(nil;\ldots;assign(\cdot,\cdot);nil;\ldots;nil,s) \equiv false$.

B. Fuzzy Inference in Golog

In order to interpret fuzzy conditionals we need to define the truth value of these conditionals. For that, we define the truth values for the following inference rules as given by Zadeh in [15]: (1) Entailment rules, (2) Negation rules, (3) Conjunction and Disjunction rules, and (4) Categorical rules.

1) Entailment rules: The entailment rule refers to a simple entailment of the form `IF X = C THEN Y = B`$\ldots$. If some antecedent of a fuzzy rule holds, i.e. the control variable `X`'s value belongs to a certain category `C`, then the consequence of the rule is applied by means of rule 4 (see below). If $\phi = is(\mathfrak{f},c)$, then

$$\phi \equiv is(\mathfrak{f}[s],c).$$

As we already defined the value-membership relation in the situation calculus (cf. Def. 1), the above definition is straightforward.

2) Negation rules: In fuzzy control there are rules which refer to an object not being part of a certain category. Usually, they are written in the form `IF X` $\neq$ `C THEN Y = B`$\ldots$, i.e. if $\phi = is_{\mathtt{C}}(\mathfrak{f},c)$ then

$$\phi \equiv is_{\mathtt{C}}(\mathfrak{f}[s],c).$$

3) Conjunction and Disjunction rules: Fuzzy variables are connected disjunctively and conjunctively, resp., as in `IF X` $= \mathtt{C}_1 \circ \cdots \circ$ `X` $= \mathtt{C}_n$ `THEN Y = B`$\ldots$ where $\circ \in \{\star, \oplus\}$ refers to the t-norm and s-norm, resp. Hence, if $\phi = is_\circ(\mathfrak{f}, c_1, \ldots, c_n)$ then

$$\phi \equiv is_\circ(\mathfrak{f}[s], c_1, \ldots, , c_n)$$

referring to $is_\star$ and $is_\oplus$ as given in Def. 1. As the fuzzy conditions ϕ and ψ are logical formulae defined by the predicate is (Def. 1), also $\phi \wedge \psi$, $\phi \vee \psi$, and $\neg\phi$ are formulae and we can compose complex fuzzy conditions.

4) Categorical rules: Categorical rules are rules stating properties of fuzzy variables, like X *is small*. In our Golog dialect, this simply breaks down to assigning a certain category to a fuzzy fluent. We do this by making use of the situation calculus action $assign$ as defined in [5]:

For a fuzzy fluent $\mathfrak{f}(\vec{x},s)$ we define a special assignment operator $assign$ which assigns the value of a category c to the fluent $\mathfrak{f}$ in situation s. The intention is to assign the category's mean value $\hat{u}$ with

$$\begin{aligned} cog(c) = \hat{u} \equiv \exists u_0,\ldots,u_k,\mu_{u_0},\ldots,\mu_{u_k}.&\mathfrak{F}(c,u_0,\mu_{u_0}) \wedge \cdots \wedge \\ &\mathfrak{F}(c,u_k,\mu_{u_k}) \wedge u_0 \neq \cdots \neq u_k \wedge \forall u^*,\mu^*.(u^* \neq u_0 \wedge \cdots \wedge \\ &u^* \neq u_k \wedge \mu^* \neq \mu_{u_0} \wedge \cdots \wedge \mu^* \neq \mu_{u_k} \supset \neg\mathfrak{F}(c,u^*,\mu^*)) \wedge \\ &\hat{u} = \sum_{i=1}^{k} u_i \cdot \mu_{u_i} \Big/ \sum_{i=1}^{k} \mu_{u_i} \end{aligned} \tag{5}$$

denoting the centre of gravity of all values defining the category c.[1] The assignment action can be formalised by adding the following case to the successor state axiom of fluent $\mathfrak{f}$: $\mathfrak{f}(\vec{x},do(a,s)) = \hat{u} \equiv \ldots a = assign(\mathfrak{f},c) \wedge cog(c) = \hat{u}\ldots$. The assignment operator is a special primitive situation calculus action.

C. Hedges

As another useful feature for dealing with linguistic terms, we introduce so-called hedges. Hedges are linguistic modifiers that are acting on a fuzzy set membership function in order to modify its meaning. A common example is the hedge *very*. If *weak pressure* is a fuzzy set, then *very weak pressure*, or *extremely weak pressure* are examples for hedges. One can distinguish between several kinds of hedges: (1) Concentration, (2) Dilatation, and (3) Artificial Hedges.

The idea of concentration is to express the cumulativeness of a membership function. For example, we would like to express that something is *very small*, or *very very large*. This is done by means of concentration, and we introduce some new function into the situation calculus for this purpose.

Definition 2 *Let $\mathfrak{f}(\vec{x},s)$ be a fuzzy fluent with normalised support $\mathfrak{F}(c,u,\mu)$.*[2] *(1) Concentration: we define a function $very : \mathbb{R} \to \mathbb{R}$ with $very(\mu_c) = \mu_c^2$; (2) Dilatation: we*

[1] Note that we have chosen the centre of gravity as a defuzzifier here, every other defuzzifier function is applicable as well and can be selected.

[2] With normalised support we mean that the values of $\mathfrak{F}$ range between 0 and 1.

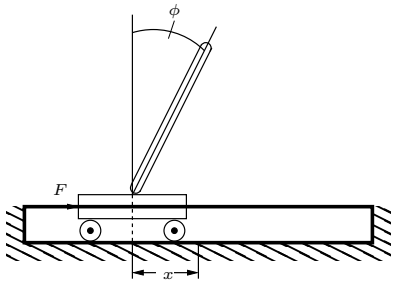

Fig. 2. The Pole Balance Task

define a function $less : \mathbb{R} \rightarrow \mathbb{R}$ *with* $less(\mu_c) = \mu_c^{\frac{1}{2}}$*; and (3) Artificial Hedges: We define a function* $plus : \mathbb{R} \rightarrow \mathbb{R}$ *and a function* $minus : \mathbb{R} \rightarrow \mathbb{R}$ *with* $plus_c(u) = (\mu(u))^{1.25}$ *and* $minus_c(u) = (\mu(u))^{0.75}$.

IV. A First Fuzzy Control Example in Golog

A. The Pole Balancing Domain

The pole-balancing problem requires the proposal of a close-loop feedback control system with the desired behaviour of balancing a pole (an inverted pendulum) that is connected to a motor driven cart by a ball-bearing pivot. The movement of the cart is restricted to the horizontal axis by a track, and the pole is free to move about the horizontal axis of the pivot. The state of the system is defined by four real values: the angle of the pole ϕ, the angular velocity of the pole $\dot{\phi}$, the position of the cart relative to the centre of the track x and the velocity of the cart $\dot{x}$. The output of the control system is a forward or backward movement of the cart in form of a force applied to it (see Fig. 2 and Tab. I).

TABLE I
The Control Parameter

Symbol	Name	Description
ϕ	Pole angle [rad]	current angle of the pole
$\dot{\phi}$	Pole velocity [rad/sec]	angular velocity of the pole
$\ddot{\phi}$	Pole acceleration [rad/sec^2]	ang. acceleration of the pole
x	Cart position [m]	measured as relative offset from the middle of the track
$\dot{x}$	Cart velocity [m/sec]	current velocity of the cart
$\ddot{x}$	Cart acceleration [m/sec^2]	current acceleration of the cart
m_c	Mass of the cart, $m_c = 1$ kg	$\div$
m_p	Mass of the pole, $m_p = 0.1$ kg	$\div$
l	Length of the pole, $l = 1$ m	$\div$
t	Time [sec]	$\div$
F	force [N]	force applied to the cart in steps of (± 10 N)
h	track limit, ± 2.5 m	border of the track
r	pole failure angle [rad]	$r \in [-0.209, 0.209]$, ($\pm 12^\circ$ to 0°)
τ	time step	discrete time step

We connected a simple pole balance simulator to our Golog run-time system for providing the closed control loop and for applying the force to the cart appropriately. In our simulator, the cart behaves according to the following motion equations:

$$\ddot{\phi}_t = \frac{g \sin\phi_t + \cos\phi_t \left(\frac{-F_t - m_p l \ddot{\phi}_t^2 \sin\phi_t}{m_c + m_p} \right)}{l \left(\frac{4}{3} - \frac{m_p \cos^2 \phi_t}{m_c + m_p} \right)}$$

$$\ddot{x}_t = \frac{F_t + m_p l \left(\phi_t^2 \sin\phi_t - \ddot{\phi}_t \cos\phi_t \right)}{m_c + m_p}$$

As we show in the next section, the inputs ϕ and $\dot{\phi}$ are connected to our system as exogenous fluents, the force is applied to the cart via primitive actions.

B. Controlling the Cart with Golog

Axiomatising fuzzy controllers in Golog requires a basic action theory. For our axiomatisation of fuzzy controllers we need to add sentences defining linguistic categories, exogenous fluents, fuzzy fluents, and membership functions. We add these sentences to the set $\mathcal{D}_{S_0}$, defining what is true in the initial situation.

$$\begin{aligned}
\mathcal{D}_{S_0} = \{ & \ldots, \mathit{ffluent}(\phi), \mathit{ffluent}(\dot{\phi}), \mathit{ffluent}(F), \\
& \mathit{category}(med_{\phi}^{-}), \mathit{category}(small_{\dot{\phi}}^{-}), \mathit{category}(zero_{\phi}), \\
& \mathit{category}(med_{\dot{\phi}}^{-}), \ldots, \mathit{category}(med_{F}^{-}), \ldots \\
& \mathfrak{F}(\phi, med_{\phi}^{-}, \ldots, (-\pi/2, 1.0), \ldots, (\pi/2, 0.04)), \\
& \mathfrak{F}(\phi, small_{\dot{\phi}}^{-}, \ldots), \ldots, \\
& \mathfrak{F}(\dot{\phi}, \ldots), \mathfrak{F}(F, \ldots) \\
& \mathit{connect}(\phi, \mathit{cart_ang}), \mathit{connect}(\dot{\phi}, \mathit{cart_ang_vel}) \\
& \ldots \}
\end{aligned}$$

Above, we defined the fuzzy fluents ϕ, $\dot{\phi}$ and F together with their fuzzy sets $\mathfrak{F}$. For each fluent, denoted by the predicate *category*, we define the value/membership pairs for each fluent. We defined the categories *small*, *medium*, *zero* for each fuzzy fluent (indicated by the subscript), for positive and negative values (indicated by the superscript). Finally, we need to pair the fuzzy fluent with an exogenous fluent, which will be needed for updating the quantitative fluent value. Here, *cart_ang*, *cart_ang_vel*, and *cart_force* are such exogenous fluents which relate to the fuzzy fluents ϕ and $\dot{\phi}$. Note that while in our formalisation we refer only to discrete fuzzy sets which explicitly enumerate the value-membership relationship for every fluent value, in our Prolog implementation we make use of triangular-shaped membership functions for the cart controller, though.

Now, we can define the controller and the rule base in Golog. The fuzzy controller for the inverted pendulum is simply the set of rules based on the qualitative categories of the sensor input ϕ, the angle of the pendulum, and its angular velocity $\dot{\phi}$. The control output is the force F applied to the cart. The Golog procedure for balancing the pole and the corresponding rule base is given below. As long as the reference input $r(t)$ is not exceeded, i.e. the pendulum's angle is not beyond ± 0.209 radians, we update our sensor values, consult the rule base, and finally apply the force to the cart.

proc pole_balance
  while | $cart\_ang$ | ≤ 0.209 **do**
    $update$; $rulebase$; $apply\_force$
  endwhile
endproc

proc rulebase
  fuzzy_controller(...;
  if is$_\star(\phi, zero_\phi, \dot{\phi}, med^{+}_{\phi})$ **then** $assign(F, med^{-}_{F})$;
  if is$_\star(\phi, small^{+}_{\phi}, \dot{\phi}, small^{-}_{\dot{\phi}})$ **then** $assign(F, zero_F)$;
  ...; **default**$(assign(F, zero_F))$
) /* end fuzzy_controller */
endproc

For the whole set of rules used in the pole balancing domain, we refer to the literature, e.g. [11].

As already mentioned, the rule base consists of a block of rules enclosed by the program statement **fuzzy_controller**(). The rules themselves are arranged as sequences of conditionals, with a fuzzy condition as antecedent and an assignment action as consequence. In our controller implementation we moreover make use of the default assignment, which we introduced in the previous section. When the control flow returns from the rule base, we have at least one assignment to the output force.

Finally, we need to apply the force to the cart.

proc apply_force
  if $F = med^{-}_{F}$ **then** push_left_strong
    else if $F = small^{-}_{F}$ **then** push_left
    ...
    else $F = med^{+}_{F}$ **then** push_right
  endif
endproc

V. Conclusion

In this paper we presented an approach to integrate fuzzy controllers into the high-level robot planning and programming language Golog. In an earlier paper, a semantics for qualitative situation calculus fluents was presented, which was based on fuzzy sets. In this paper we make use of these qualitative fluents and show an embedding of fuzzy control rules in the logic-based high-level language Golog. We introduce a new program statement which allows for formulating fuzzy rules in Golog programs and define its Golog transition semantics. For evaluating the antecedent of fuzzy rules we define the truth values of these formulae in the situation calculus. The consequence of a fuzzy rule is simply mapped to a special primitive Golog action which defuzzifies the output value and assigns it to a fuzzy fluent. Finally, we defined hedges like *very* or *less* which are very useful when formulating qualitative action theories.

We implemented a pole balancing agent in Golog using fuzzy controller as presented in this paper. With a simple set of 25 rules the agent was able to keep the pole upright not taking longer than $4ms$ for any of its decisions. Thus, as a proof of concept, we were able to combine the reactive decision making process provided by a fuzzy controller with the expressive power of the Golog language. While the experimental results mentioned in this paper are clearly preliminary and can only be seen as a proof of concept, we aim at tackling more interesting control problems in the future. Of course, a toy domain like the pole balancing domain, which is a prototypical problem for applying fuzzy rules is not challenging. However, we think that the approach to combine purely reactive control patterns like fuzzy rules with a deliberative reasoning engine, which comes with the situation calculus and Golog, is intriguing and can be beneficially deployed in the robotics context for the future. The possibility to use such qualitative control laws in Golog increases its expressiveness and will increase the possibilities to tackle robotic high-level problems.

Acknowledgments

This work was partly supported by the German National Science Foundation (DFG) in the Priority Program 1125, *Cooperating Teams of Mobile Robots in Dynamic Environments*. A. Ferrein is currently funded by the Alexander von Humboldt Foundation in the Feodor Lynen programme. We would like to thank the anonymous reviewers for their helpful comments.

References

[1] G. De Giacomo, Y. Lésperance, and H. J. Levesque. ConGolog, A concurrent programming language based on situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.
[2] G. De Giacomo, H. Levesque, and S. Sardiña. Incremental execution of guarded theories. *Computational Logic*, 2(4):495–525, 2001.
[3] D. Dubois and H. Prade. An introduction to fuzzy systems. *Clinica Chimica Acta*, 270(1):3–29, 1998.
[4] A. Ferrein and G. Lakemeyer. Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems, Special Issue on Semantic Knowledge in Robotics*, 56(11):980–991, 2008.
[5] A. Ferrein, S. Schiffer, and G. Lakemeyer. A fuzzy set semantics for qualitative fluents in the situation calculus. In *Proceedings of the International Conference on Intelligent Robotics and Applications*, Lecture Notes in Computer Science. Springer Verlag, 2008. to appear.
[6] R. Isermann. On fuzzy logic applications for automatic control, supervision, and fault diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 28(2):221–235, 1998.
[7] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. *J. of Log. Progr.*, 31(1-3), 1997.
[8] H. Liu. A fuzzy qualitative framework for connecting robo qualitative and quantitative representations. *IEEE Transactions on Fuzzy Systems*, 16(6):1522–1530, 2009.
[9] H. Liu, D. J. Brown, and G. M. Coghill. Fuzzy qualitative robot kinematics. *IEEE Transactions on Fuzzy Systems*, 16(3):808–822, 2008.
[10] J. Mendel. Fuzzy logic systems for engineering: a tutorial. *Proceedings of the IEEE*, 83(3):345–377, 1995.
[11] M. Passino and S. Yurkovich. *Fuzzy Control*. Addison-Wesley-Longman, 1998.
[12] F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46(3):325–361, 1999.
[13] R. Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.
[14] A. Saffiotti. Fuzzy logic in autonomous robotics: behavior coordination. In *Proc. IEEE Int. Conf. on Fuzzy Systems*. IEEE Computer Society Press, 1997.
[15] L. Zadeh. Knowledge representation in fuzzy logic. *IEEE Transaction on Knowledge and Data Engineering*, 1(1), 1989.