Non-monotonic Reasoning on Board a Sony AIBO

Author

Billington, David, Estivill-Castro, Vladimir, Hexel, Rene, Rock, Andrew

Published

2007

Book Title

Robotic Soccer

DOI

10.5772/5124

3

Non-monotonic Reasoning on Board a Sony AIBO

David Billington, Vladimir Estivill-Castro, René Hexel, and Andrew Rock
Griffith University
Australia

1. Introduction

Robots are today a reality. Moreover, robots have moved from assembly lines to being around human beings. Mobile autonomous robots are now a common sight in Korean airports. Other notable examples are LEGO's Mindstorms and Spybotics, who not only have a massive penetration in the toy market, but have penetrated the research and academic environment (Wallich, 2001). Robots are also being sold commercially as companions, or used as museum guides (Thrun *et al.*, 1999), and even as the long awaited vacuum cleaner (Kahney, 2003). The expectation that robots would be around us inspired Isaac Asimov to write "I Robot" as part of a series of books and to develop the character Susan Calvin who enunciated the Three Laws of Robotics:

1. A robot may not injure a human being, or, through inaction, allow a human to come to harm.
2. A robot must obey orders given to him by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

To follow these rules, a robot would need to reason about actions and their potential effect. Reasoning is a fundamental capability of intelligent systems and much progress has been made (Marek & Truszczynski, 1993; Rich & Knight, 1990) (this is also illustrated by the 4 chapters dedicated to uncertain knowledge and reasoning in (Russell & Norvig, 2002), at present the most widely accepted textbook in Artificial Intelligence and 57th most cited computer science publication ever). Most notably, for intelligent and robotic systems it is essential that such reasoning be capable of withdrawing some conclusion in the light of new evidence (including the negation of what used to be considered a fact). This is called non-monotonic reasoning.

This chapter will describe how a Sony AIBO performed on board non-monotonic inferences on two settings present on the RoboCup competition (Veloso *et al.*, 1998). Robotic soccer is the most challenging endeavor from the perspective of multi-agent technology (Wooldridge,

Source: Robotic Soccer, Book edited by: Pedro Lima, ISBN 978-3-902613-21-9,
pp. 598, December 2007, Itech Education and Publishing, Vienna, Austria

2002). We argue that non-monotonic reasoning is useful in even the dynamic,[1] inaccessible,[2] adversarial and non-deterministic[3] environment of robotic soccer, where reactive systems have received much attention. We also present an example for the RoboCup@Home setting. We achieve this common sense behavior by an implementation of Plausible Logic (and some algorithm fine-tuning) in C++.

We will commence the discussion by applying Plausible Logic inferences to make sense of the sightings in the configuration of the 2005/2006 4-legged league field of play. In particular, we show that we can analyze the objects reported by the vision module in a frame (or in a sequence of frames) and determine which were phantom sightings and which could actually be valid sightings. This assists localization as it dynamically selects proper inputs (landmarks). Localization means that the software in the robot must gather information from its sensors and arrive at a reasonable (and accurate) conclusion about its location and orientation. We argue that there is a role to be played by reasoning when localizing, with a loose analogy to when people use previous knowledge to explore an environment they have some information about. Our problem is different from a pure SLAM problem. Here, we assume some previous knowledge of the environment, so we can reason about and contrast our observations with our prior knowledge.

There are many algorithms to deal with the error in odometry as well as errors in other inputs for localization (vision or laser sensors). These usually fall into three main categories (the family of Kalman Filters (KF), the family of Markov Models (MM) and the family of Monte Carlo (MCL) localizers). We do not advocate their elimination. In fact, we use the Monte Carlo Localization approach for localizing Sony AIBO robots in the 4-legged league of RoboCup. However, we introduce non-monotonic reasoning as a filter before the localization process takes observations as inputs.

We believe our approach offers an alternative to the problem of *data fusion*. For data fusion, probabilistic models are favored over reasoning with logic models. For example, for combining information from several sources, their reliability is modeled using probabilities and "reasoning with uncertainty" is performed using general models that include applications to sensor fusion (Haemmi & Hartmann, 2006, and references therein). As sensors become more sophisticated, and as entire modules on board a robot collect information about the environment, reasoning is essential to integrate and comprehend such sources of information (some could come from observations by teammates). Non-monotonic reasoning would be most effective when the information is contradictory. In fact, contradictory information is actually the common case. At the sensor level, an odometry sensor will usually be in disagreement with the distance computed by using projective geometry and trigonometric equations from the images of a digital camera, and these will also exhibit differences with an infrared-range sensor. Information from teammates has a lag in time.

---

[1]The environment is dynamic if it will evolve in the time gap between the sensors collecting information and the agent performing an action (Wooldridge, 2002).

[2]Information about the entire environment may not be possible to collect (Wooldridge, 2002).

[3]An environment is non-deterministic if an action may not have the expected outcome (Wooldridge, 2002), like a skid because the surface is smoother than anticipated.

We propose to use non-monotonic reasoning to accept the inconsistent information and resolve it to obtain the most plausible interpretation of the state of a robot and its environment. Identifying this state is clearly a crucial initial step towards making a decision and then acting. We also want some assessment of the likelihood of that state for the decision making process. The influential works by Brooks (Brooks, 1991) have lessened the interest in using symbolic/logic approaches. We argue here that a computable non-monotonic logic has a role to play.

Our systems have been implemented and operated by the Mi-PAL team in RoboCup (2005, 2006, and 2007). The current robotic platform is the Sony AIBO robot.

2. Background on Plausible Logic

Non-monotonic reasoning (Antoniou, 1997) is the capacity to make inferences from a database of beliefs and to correct those as new information arrives that make previous conclusions invalid. Although several non-monotonic formalisms have been proposed (Antoniou, 1997), *Plausible Logic* (PL) (Billington & Rock, 2001; Rock & Billington, 2000) is currently the only one with an efficient non-looping algorithm (Billington, 2005). Another very important aspect of PL is that it distinguishes between formulas proved using only factual information and those using plausible information. PL allows formulas to be proved using a variety of algorithms, each providing a certain degree of trust in the conclusion. Because PL uses different algorithms, it can handle a closed world assumption (where not telling a fact implies the fact is false) as well as the open world assumption in which not being told a fact means that nothing is known about that fact. The β algorithm for PL uses the closed world assumption while the π algorithm uses the open world assumption.

If only factual information is used, PL essentially becomes classical propositional logic. However, when determining the provability[4] of a formula, the algorithms in PL can deliver three values (that is, it is a three-valued logic). The proving algorithms terminate assigning the value +1 to a formula that has been proved and -1 to a formula that has been disproved. It assigns the value 0 when the formula cannot be proved and attempting so would cause infinite recursive looping.

In PL all information is represented by three kinds of rules and a priority relation between those rules. Strict rules, denoted by the strict arrow → and used to model facts that are certain. For a rule $A \rightarrow l$ we should understand that if all literals in A are proved then we can deduce l (this is simply ordinary implication). A situation like *Humans are mammals* will be encoded as $human(x) \rightarrow mammal(x)$.

Plausible rules $A \Rightarrow l$ use the plausible arrow ⇒ to represent a plausible situation. If we have no evidence against l, then A is sufficient evidence for concluding l. For example, we write *Birds usually fly* as $bird(x) \Rightarrow fly(x)$. The intent is to record that when we find a bird we may conclude that it flies unless there is evidence that it may not fly (like knowing it is a penguin).

Defeater rules $A \sim\!> \neg l$ mean that if A is not disproved, then it is too risky to conclude l. An example is *Sick birds might not fly* which is encoded as $\{sick(x), bird(x)\} \sim\!> \neg fly(x)$. Defeater

[4]Provability here means determining if the formula can be verified/proved.

rules prevent conclusions that would otherwise be too risky. This could happen in a chain of conclusions from plausible rules.

Finally, a priority relation > between rules $R_1 > R_2$ indicates that R_1 should be used instead of R_2. In this chapter we actually demonstrate the expressive power of this aspect of the formalism. For example from

$$\begin{aligned} \{\} &\rightarrow quail(Quin) && \textit{Quin is a quail} \\ quail(x) &\rightarrow bird(x) && \textit{Quails are birds} \\ R_1{:}bird(x) &\Rightarrow fly(x) && \textit{Birds usually fly} \end{aligned}$$

one would logically accept that *Quin* usually flies. From the knowledge base

$$\begin{aligned} \{\} &\rightarrow quail(Quin) && \textit{Quin is a quail} \\ quail(x) &\rightarrow bird(x) && \textit{Quails are birds} \\ R_2{:}quail(x) &\Rightarrow \neg fly(x) && \textit{Quails usually do not fly} \end{aligned}$$

we would reach the correct conclusion that Quin usually does not fly. But what if both knowledge bases are correct, that is both rules R_1 and R_2 are valid. We see that R_2 is more specific than R_1 and so we add $R_1 > R_2$ to a knowledge base representing the beliefs of a robot that knows both. Then PL allows the agent to reach the proper conclusion that Quin usually does not fly, while if it finds another bird that is not a quail, the agent would accept that it flies.

Note that the Three Laws of Robotics are an example of how humans describe a model. They define a general rule, and the next rule is a refinement. Further rules down the list continue to polish the description. This style of development is not only natural, but allows incremental refinement. Indeed, the knowledge elicitation mechanism known as *Ripple Down Rules* (Compton & Jansen, 1990) extracts knowledge from human experts by refining a previous model by identifying the rule that needs to be expanded by detailing it more. The models presented here are each a progressive refinement of the previous one.

3. Plausible Logic for Localization

Non-monotonic reasoning has long been considered too complex for real-time environments. Visual robot localization in the 4-legged league places particularly stringent demands on the processor. Video camera systems typically operate at a rate of 30 frames per second, which allows only about 30 ms to perform full image recognition, feature extraction, and consistency verification. Moreover, poor lighting conditions can make color calibration extremely difficult. More often than not, vision systems make errors in object recognition (in particular, they may occasionally miss the landmarks for localization or report non-existent objects as visible). In this section we present our first application of PL. We use it to make sense of the sightings in a scene before they are used as landmarks for localization.

3.1 The Problem

The most well known family of techniques to interpret the input provided by a sensor with some noise is derived from the *Kalman Filter* after the 1960's publication by R.E. Kalman describing a recursive solution to the discrete-data linear filtering problem. Since that time, due in large part to advances in digital computing, the Kalman Filter has been the subject of

extensive research and application, particularly in the area of autonomous or assisted navigation.

In robot localization, alternative techniques have emerged. Most notably, grid-based Markov localization and Monte Carlo localization (Fox *et al.*, 1999; Gutmann & Fox, 2002; Thrun *et al.*, 2001). These techniques are based on a paradigm that still uses probability distributions. The manipulation of the probabilistic representation is slightly different across these schemes. While the Kalman filter (KF) uses some mathematically defined probabilistic model (usually multivariate Gaussian distributions), the Markov model (MM) represents a distribution as a histogram (one could say Kalman is using parametric statistics while Markov localization uses non-parametric statistics). On the other hand, Monte Carlo localization (MCL) represents the probability distribution by a population of weighted samples (also close to a non-parametric model of the distribution) but rather than representing the distribution by piece-wise values on a grid, Monte Carlo uses a population of cases. Fundamentally, the three approaches update the current belief using Bayes theorem to incorporate the knowledge from a sensor and to update the current belief. They use conditional probabilities to represent the prior knowledge and posterior knowledge of the state of the world. In an autonomous mobile agent, the belief is revised by an observation as well as by an action. While the non-parametric schemes seem better equipped to deal with some of the performance issues of Kalman filters, and resolve some data fusion issues, they still are not able to rule out inconsistencies. For example, a phantom object in a frame can create a bump in the distribution that will be removed after many new consistent observations.

In particular, an example is a frame where the vision module reports two objects as visible, even though it is not possible for these objects to appear together in the same frame. In the case of Robotic soccer for the 4-legged league, this would be illustrated by the vision system seeing the opponent's goal as well as their own goal within the same frame. The localization approaches need to estimate $Prob(\text{visible scene} \mid \overrightarrow{pos})$, where the visible scene is a description of all visible objects, and vector $\overrightarrow{pos}$ is the current belief. To avoid describing probabilities for all possible scenes, one approach is to regard some observations as independent and modify the current belief by the product of $Prob(See\_front\_goal \mid \overrightarrow{pos})$ and $Prob(See\_back\_goal \mid \overrightarrow{pos})$ (for example when seeing both goals). The problem with this is that because $\overrightarrow{pos}$, has significant error regarding the orientation and pan of the head of the Sony AIBO, both of these probabilities are unlikely to be zero in any reasonable sensor model. This would result in creating a local mode in the probability distribution (in Markov and Monte-Carlo models) while creating a significant enlargement of the covariance matrix for the spatial Kalman filter. In fact, we know it is impossible to see both goals in the same frame, that is, we know $Prob(See\_front\_goal \wedge See\_back\_goal \mid \overrightarrow{pos}) = 0$, for all postures $\overrightarrow{pos}$. However, we just indicated that representing $Prob(See\_front\_goal \wedge See\_back\_goal \mid \overrightarrow{pos})$ as a function of $Prob(See\_front\_goal \mid \overrightarrow{pos})$ and $Prob(See\_back\_goal \mid \overrightarrow{pos})$ is rather complicated.

So the alternative is to generate a database of cases for these situations where domain knowledge allows us to plug in suitable values. The problem with this approach is that these cases become not only a few, but a rather large number. It then becomes hard to

ensure that this database of facts is accurate (or complete, or consistent). Furthermore, we must ensure that we are using this database to rule out observations at the right time. Because of the modeling assumptions in localization algorithms, it is important that the observations from sensors be as reliable as possible (otherwise, the convergence is too slow or the artifacts to handle the kidnap problem introduce other high modes in the representation of the distribution). In RoboCup most teams participating in the competition perform rules of thumb that fall in the realm of classical logic (sensible sanity checks); that is, they will filter out observations from the vision system that indicate that opposite goals were seen in the same frame. These *inconsistent*[5] inputs are simply, partially or not at all, used for localization.

The situation becomes difficult to manage, as entangled with the localization code is a series of logical tests that check special cases. Some code filters the observations that are considered inconsistent. This *consistency* module rapidly becomes a large piece of software, hard to verify for correctness or completeness. Our first thesis is that such a filter of inconsistent observations is better handled by some logic. The second thesis is that such a logic should not only be capable of ruling out observations, but allow reasoning about them to provide informative inputs to the localization module.

We have experimented with other alternatives (Billington *et al.*, 2005) to model the field of RoboCup 2005/2006[6] for the 4-legged league. These usually result in a complex description of the potential inconsistent inputs. In particular, it is very likely that most imperative object-oriented or procedural (in the case of the Sony AIBO C++) implementations of this, will result in at least incomplete models, and more seriously, deliver inconsistent models as they are usually developed incrementally as deeper and deeper nesting of `if-then-else` statements. Our analysis reflects that also some logic approaches rapidly result in a large number of rules. We believe most competing teams in RoboCup do not have a complete set of rules for handling, for example when vision detects four landmarks in a frame two of which are phantoms (blobs from the audience or off-field objects which fit the landmark characteristics but appear to vision as landmarks because of calibration or very similar color). Most teams survive this because these cases of many phantoms in one frame are reasonably rare in the constrained environment of labs or competition venues. However, they do pose a very serious threat to the correctness of their overall play (moreover, such faults become extremely difficult to reproduce and detect). The point we are making is that even from the software engineering, software verification and validation point of view, we need a complete and correct logic theory of the consistency of the vision reports.

Our example here analyzes the challenge of imperfect vision reports. That is, in a single frame, the analysis of an image may actually perceive two blobs of yellow color and one of blue that are rectangular enough for all of them to be considered as goals. Again, any software/logic that rules out two rectangular blobs of yellow, perhaps on the basis that one is larger than the other, or one is above the field of vision, or one is next to green, is performing some reasoning based on domain knowledge. What we are arguing here is that if all those ways of ruling out sightings of landmarks are not concentrated in a single place

[5]We use here the word *inconsistent* for an observation that is in some way in contradiction to what we expect from our knowledge of the domain. Note that we will use inconsistent theory or incomplete theory in the usual sense used by logicians.

[6]Previous versions of the field are very similar.

represented in logic, then the software is very likely to have such rules in several modules, resulting in high coupling of these, and more seriously, in incomplete and inconsistent modeling of the reasons why some sightings are ruled out before they are used for localization. As the robots move to more realistic environments more reasoning is needed.

3.2 Modeling with Plausible Logic

We introduce the modeling of consistent sightings incrementally. We start with a simple example but the point is not only to help the understanding of using non-monotonic logic, but to illustrate that in PL higher-level models are introduced incrementally as extensions of the previous model. This process allows us to model the most important (and most likely) scenarios upfront, while refining the models to handle the complexity of more specialized and sophisticated cases later. The execution of PL proofs on the AIBO is abstracted so that upgrading the model does not represent reprogramming the C++ code. The implementation of PL not only provides the algorithms for obtaining proofs, but provides a logic programming language DPL (Rock) for presenting facts, and describing a theory.

3.2.1 Model 1

We first need to represent the domain knowledge. Each 2005/2006 4-legged league soccer field has fundamentally 6 landmarks for localization. These are two goals (one yellow and the other blue) and four posts. Each post has two colors, and pink is always one of these. The two posts near the blue goal have blue as one of the colors while the two posts on the yellow side have yellow. This color-coding allows the identification of landmarks for a robot as Front Goal (FG), Back Goal (BG), Left Post (LP), Right Post (RP), Right Back Post (RBP) and Left Back Post (LBP). We also take advantage of the fact that although in 2005 the field has been enlarged, there are still some scenes that can be ruled out. The horizontal angle of view is 56.9°, but to simultaneously see LP and RBP would require a view greater that 67.5°.

First, the facts about the world are presented by type declarations in the logic programming language as follows. There are two goals.

```
type GoalType = {FG, BG}.
```

There are four posts.

```
type PostType = {LP, RP, RBP, LBP}.
```

A landmark is either a goal or a post, that is $Landmark=GoalType \cup PostType$. In the programming language we have

```
type Landmark = GoalType + PostType.
```

The next step is to define the inputs as predicates. In general, any piece of information we can retrieve from sensors or messages from teammates can be modeled as an input. Each input is introduced in the logic model as an axiom and these inputs trigger (fire) the plausible assumptions that appear in the model description. It is cumbersome to write, for every axiom a, the two plausible rules $a \Rightarrow p$ and $\neg a \Rightarrow \neg p$, where p is the plausible assumption to be fired by a. A macro of DPL simplifies this.

```
$=declareInput$(a$,p$)${$#
 input{$+a}. {$+a} => $+p. {~$+a} => ~($+p). $}$#
```

Now, we can write the plausible assumptions (this is what vision reports). First, *See*(*x*) will evaluate to true if and only if the vision module reports a sighting of landmark *x*.

```
type See(x <- Landmark).
```

This first model provides correct results only if the vision module reports exactly one landmark or none. Now, we are in a position to describe consistency rules. By default, when vision does not report a landmark, we do not forward anything to localization. This is an easy default case. This is R_1:{} ⇒¬*Cs*(*x*) while in the programming language DPL we write

```
R1: => ~Cs(x).
```

However, if vision reports a landmark, we believe it; since for only one frame we have no other information to rule this out. PL writes this as R_2: *See*(*x*)⇒*Cs*(*x*) R_2>R_1. Note the relationship between rules. Now, the DPL equivalent is

```
R2: See(x) => Cs(x).  R2 > R1.
```

This works because we also provide a statement that any frame that has two or more landmarks should be ignored. This completes the programming of this simple model.

3.2.2 The Path to Implementation

Initially, the only implementation available of PL was in Haskell, and it was unclear that this implementation, even if translated to C++ would be fast enough to operate in the time slot for processing a vision frame (which is the usual time slot for doing all computation without losing frames, and thus possibly even losing sightings of critical objects like the ball). While originally (Billington *et al.*, 2005, 2006) we enabled PL without running the inference engine on the Sony AIBO, it became clear that this had computability limitations. This chapter focuses only on the scenario where the inference engine is operating on board the Sony AIBO. In order for PL proofs to be developed on board we had to extend the logic programming language DPL by adding automatic production of C++ macros and automatic production of gluing code. Also, we developed a template method and an architecture that makes model loading a component-replacement process. The PL was made to run on-board the Sony AIBO using a C implementation of the inference engine. The C implementation also ran on MAC-OS and LINUX since it used standard C-language constructs.

Extensions to DPL enable generation of C++ glue code. Namely, we took the decision we would read or evaluate sensor input only once. We would store the outcome of evaluating a plausible assumption in a C++ Boolean variable. For vision, for example, the C++ expression `FG` has value `true` iff the front goal is visible. This is an input axiom of the description that will be asserted either positively or negatively.

```
$+declareInput$("FG"$,See(FG)$)
```

Similarly, we have five more declarations. These input axioms are the atoms the logic will talk about and can be given initial values by the sensors of the robot (or the modules that read those sensors).

```
$+declareInput$("BG"$,See(BG)$)   $+declareInput$("RBP"$,See(RBP)$)
$+declareInput$("LP"$,See(LP)$)   $+declareInput$("LBP"$,See(LBP)$)
$+declareInput$("RP"$,See(RP)$)
```

An outcome of this is gluing code that declares the necessary Boolean variables in the corresponding C++ module (creates statements of the form `bool FG;`).

We proceed now to describe the special template method. The special template method consists of three phases. We already alluded to the first phase `INIT_ALL_FALSE()`. This is implemented as a macro that creates the necessary definitions of C++ Boolean variables for all input axioms. For example, we saw that the programming language enabled the declaration of an input.

```
$+declareInput$("FG"$,See(FG)$)
```

In the C++ code, there is a Boolean variable corresponding to this sensory input. In more elaborate models we will have additional inputs that indicate if one landmark is to the left of another landmark. These will also become Boolean C++ variables. Moreover, `INIT_ALL_FALSE()` not only defines those Boolean variables, but sets their values to false (C++ statements of the form `FG=false;`). This is just a default initialization. In temporal models that use information from a previous frame, the values of input axioms in one frame are copied to a corresponding frame-indexed set of C++ Boolean variables.

The second phase evaluates all input predicates (i.e. collects all information from the sensors), and stores this in the C++ Boolean variables. Its implementation is a macro `UPDATE_ALL()` that queries the values of the input axioms for the current frame. Also, if there are predicates that refer to sensor values in previous time slots, they become updated. We will say more about this when we discuss a model that analyses sightings across consecutive frames. In the current example, the macro obtains the values of input axioms from the vision module. For example, a variable for the front goal previously initialized to false may now be set to true if vision has found a front goal in the current frame (the most recent vision report will be extracted and since it reports the front goal as visible the C++ executes `FG=true;`). It also provides a pointer to such an object so other attributes about the landmark can be evaluated, e.g. its perceived size or whether it is seen to the left or right of another landmark in that frame.

The last phase of the template method is the invocation of the inference engine. `PLACE_CS_ALL()` will use the inference engine to evaluate the expressions for which we requested outputs. When we are filtering localization landmarks, if a landmark is found to be consistent (evaluates to true), the information on the landmark sighting will be forwarded to the localization module (or any other module that may benefit from it, such as the behaviour to kick when the front goal is visible). In particular `PLACE_CS_ALL()` will have as many `if` statements as landmarks, each with an expression that is a call to the inference engine. For any term in the model we want to ask its value, we can make a call to the inference engine (for example, is the sighting of the front goal consistent?). The C

implementation returns one of the three values of PL. For those landmarks found consistent, the information on them is forwarded to localization.

For the analysis of each frame, we have a module named `Consistency`. The C++ code of the template method `Consistency::Run` is executed every time a new frame arrives. In the filtering for localization example, the vision module is reporting about landmark sightings in each frame. The template method runs and verifies such reports.

```
void  Consistency::Run()
{INIT_ALL_FALSE(); UPDATE_ALL(); PLACE_CS_ALL();}
```

The three macros in the template method have the responsibility of implementing the three phases. The three macros in the template method are defined in a file named `ConsistencyMacros.h` that provides the glue code to the Mi-Pal architecture for the soccer playing robots as well as glue code for a visual testing tool. The code in `ConsistencyMacros.h` is computer generated, and depends slightly on the model to be executed on the Sony AIBO.

For the particular case of the Model 1 just presented, testing (evaluating) if the front goal is consistent is a call to the inference engine. If the front goal was not seen in this frame, then `INIT_ALL_FALSE()` would have set the variable to false and `UPDATE_ALL()` would not have changed FG's value, so no landmark sighting is forwarded to localization. However, if the front goal was visible, `UPDATE_ALL()` would have set FG to true and the `if` statement in `PLACE_CS_ALL()` would fire (because the engine would have used the logic rules of the model to prove a consistent sighting), resulting in localization receiving the sighting information about the front goal.

3.2.3 Higher Level Models

Originally, we developed the simple Model 1 for validation of the entire concept of a non-monotonic logic implemented on a Sony AIBO. We now introduce progressively more sophisticated models. We present a model that handles the consistency cases when vision reports 0, 1, or 2 landmarks in a frame. The type declarations regarding the landmarks are the same as before, but we need to describe the domain in a bit more detail. In particular, we use *Opp*(*x*,*y*) to mean *x* is opposite *y*. Also *Opp*(*x*,*y*) if and only if *Opp*(*y*,*x*). This appears in the programming language as

```
type Opp(x <- Landmark, y <- Landmark - {x}).  default ~Opp(x, y).
Opp(FG, BG).    Opp(RP,  LBP).    Opp(RBP, LP).
Opp(BG, FG).    Opp(LBP, RP).     Opp(LP,  RBP).
```

Also, we want to define relative positioning on the soccer field. The predicate *LR*(*x*,*y*) means landmark *x* is to the left of landmark *y*, and there are only 0 or 1 landmarks between them. The following are facts about left-to-right placements.

```
type LR(x <- Landmark, y <- Landmark - {x}).  default ~LR(x, y).
LR(LP, FG).  LR(RP, BG).   LR(FG, RBP).  LR(RBP, BG).
LR(LP, RP).  LR(BG, LP).   LR(RP, RBP).  LR(RBP, LBP).
LR(FG, RP).  LR(LBP, FG).  LR(BG, LBP).  LR(LBP, LP).
```

We are now in a position to use inputs from vision (using the same macro to declare them). Declarations for *See*(*x*) and axioms relating this predicate to the C++ expression `FG` are as before. What is new in this model is that we now use that vision reports if one landmark appears to the left of another. Plausible assumption *SeeLtoR*(*x*,*y*) means vision reports seeing landmark *x* to the left of landmark *y*.

```
type SeeLtoR(x <- Landmark, y <- Landmark - {x}).
```

Also for efficiency of the computation of proofs, we can use rules that simplify the setting. A macro for an axiom such as `LP_FG` that fires *SeeLtoR*(*LP*,*FG*) allows us to not consider cases where `LP_FG` is asserted but either of `LP` or `FG` is not. In the programming language we define a macro call `$+declareSeeLtoR$(`*x*`$,`*y*`$)` that declares an input axiom *x_y*, rules to fire *SeeLtoR*(*x*,*y*), and then specifies the (possible) cases to ignore.[7]

```
$=declareSeeLtoR$(x$,y$)${$#
 $+declareInput$("$+x_$+y"$,SeeLtoR($+x,$+y)$)
  ignore {"$+x_$+y",~"$+x"}.  ignore {"$+x_$+y",~"$+y"}. $}$#
```

Then, we only need to use this macro, for all possible pairs of landmarks, for example

```
$+declareSeeLtoR$(LBP$,LP$).
```

For the new model, the first two rules are the same as before. Namely, nothing is to be forwarded to another module unless it is seen. But now we add that if vision reports two landmarks we know to be opposites, then we believe neither (irrespective of whether vision reports one to the left of the other).

```
R3: {See(x), See(y), Opp(x,y)} => ~Cs(x).  R3 > R2.
```

If vision reports two objects out of left-to-right order, then we also believe neither.

```
R4: {See(x), See(y), SeeLtoR(y,x), LR(x,y)} => ~Cs(x).
R4: {See(x), See(y), SeeLtoR(y,x), LR(x,y)} => ~Cs(y).  R4 > R2.
```

The complete rules for **Model 2** are expressed in the programming language as follows:

```
R1: => ~Cs(x).  R2: See(x) => Cs(x).  R2 > R1.
R3: {See(x),See(y), Opp(x,y)} => ~Cs(x).  R3 > R2.
R4: {See(x),See(y),SeeLtoR(y,x),LR(x,y)} => ~Cs(x).
R4: {See(x),See(y),SeeLtoR(y,x),LR(x,y)} => ~Cs(y).  R4 > R2.
```

Note the non-monotonic aspect of the model. In particular, the inference engine may reach the initial conclusion that nothing is to be forwarded to the localization module (by R_1) but then conclude that there is a landmark sighting to be forwarded (because of R_2). However, it may change that conclusion in light of R_3.

---

[7]While the `ignore` statements do not influence the plausible rules directly, they serve the purpose of declaring that the given combinations of inputs can be ignored.

If we want to add rules that use other aspects of the information from the sensors in the report from vision, we can also achieve this. We illustrate with rules to report a post over a goal, or the larger of two goals. We can now revise **Model 2** to **Model 2a** to use information on objects size or type, for example. We may want to report a post over a goal even if perceived in the wrong left-to-right order.

```
R1: => ~Cs(x).  R2: See(x) => Cs(x).  R2 > R1.
R3: {See(x),See(y), Opp(x,y)} => ~Cs(x).  R3 > R2.
R4a: {See(x),See(y),SeeLtoR(y,x),LR(x,y)} => Cs1(x,y).  R4a > R2.
R5: {Cs1(x,y),Post(x),Goal(y)} => Cs(x).
R5': {Cs1(x,y),Post(x),Goal(y)} => ~Cs(y).
R6: {Cs1(x,y),Post(x),Post(y),BigSmall(x,y)} => Cs(x).
R6': {Cs1(x,y),Post(x),Post(y),BigSmall(x,y)} => ~Cs(y).
```

The predicate *Cs*1(*x*,*y*) means only one of *x* and *y* is consistent with the domain knowledge. We now have that when two objects on the scene are a post and a goal and we have concluded one is inconsistent, then we prefer the post (because it is harder to confuse an object identified with two colours). The last rule says that the larger of two inconsistent posts is to be forwarded as input for localization since it is harder to perceive a large phantom post.

There is a case with 3 landmarks in a frame where Model 2 could be refined. To describe this new refined model we have to state some more facts about the environment. We use *Adj*(*x*,*y*,*z*) to say *x* is left of and next to *y*, and, *y* is left of and next to *z*.

```
type Adj(x <- Landmark , y <- Landmark - {x},
z <- Landmark - {x, y}).  default ~Adj(x, y, z).
Adj(LP,FG,RP). Adj(FG,RP,RBP). Adj(RP,RBP,BG).
Adj(RBP,BG,LBP). Adj(BG,LBP,LP). Adj(LBP,LP,FG).
```

Let us consider the case when we see three objects *x*, *y*, and *z*, known to be adjacent from left to right, but we see *x* on the wrong side of both *y* and *z*. While we do not need to revise our opinion about *x* being inconsistent (by R_4) the mutual consistency of *y* and *z* are grounds for overriding the conclusion by R_4. This leads to an extension that we name **Model 3**.

```
R5: {See(x),See(y),See(z),SeeLtoR(y,z),SeeLtoR(z,x),Adj(x,y,z)} => Cs(y).
R5: {See(x),See(y),See(z),SeeLtoR(y,z),SeeLtoR(z,x),Adj(x,y,z)} => Cs(z).
R5 > R4.
```

Similarly, if *z* is the one out of order, then believe *x* and *y*.

```
R5: {See(x),See(y),See(z),SeeLtoR(z,x),SeeLtoR(x,y),Adj(x,y,z)} => Cs(x).
R5: {See(x),See(y),See(z),SeeLtoR(z,x),SeeLtoR(x,y),Adj(x,y,z)} => Cs(y).
```

The rules that complete this model are as follows.

```
R6: {See(x),See(y),See(z),SeeLtoR(x,z),SeeLtoR(z,y),LR(x,y),LR(y,z),
     Opp(x,z)} => Cs(x).
R6: {See(x),See(y),See(z),SeeLtoR(x,z),SeeLtoR(z,y),LR(x,y),LR(y,z),
     Opp(x,z)} => Cs(y).
R6 > R3.  R6 > R4.
```

We omit the last model that handles even 4 landmarks in the same frame. But we believe this progression of models and the illustration of their design suffices.

3.3 A First Evaluation

The implementation of these PL models has been evaluated in two directions. The effectiveness of the approach was demonstrated in an ERS-7 Sony AIBO in the lab and in the actual RoboCup competitions. We evaluated the results on the robot with a `telnet` connection that displays the ID of the objects reported by vision (see Figures 1 and 2).

Figure 1 creates a scene where a non-moving Sony AIBO would have a vision module where the left post is correct, but the goal and right post are inverted. We can see in the figure the telnet connection that portrays the image captured in the robot as well as the text that reports which landmark sightings are being forwarded for localization. For Figure 1, Model 2 provides the correct result. Namely, we can forward to localization the left post, but neither the goal nor the right post should be forwarded to the localization module.

We believe it is remarkable that the PL description for Model 2 (which only aims at writing rules for frames with zero, one or two landmark sightings) obtains correct conclusions for many settings in which three sightings or more occur. This again reflects the power of modeling with PL as opposed to modeling without it. The PL is analyzing the pairs within the triplet in sight. And while two of the pairs are consistent (those involving the left post), the pair involving the goal and the right post indicates both of these are inconsistent.

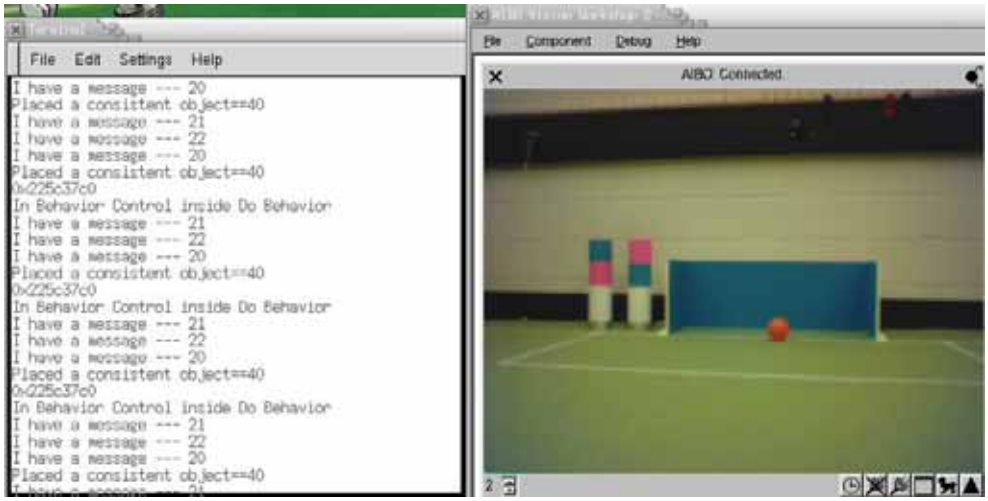


Fig. 1. The left post is correct, but the goal and right post are inverted

Not all settings with 3 landmarks are correctly identified with Model 2. Figure 2 is a setting where Model 2 rules all landmark sightings as inconsistent. In this case, all objects are involved with another object to form a pair seen in the incorrect left to right order with respect to the domain knowledge. While Model 2 rules this setting as inconsistent Model 3 correctly identifies that the left post and the goal constitute a pair seen in the expected order and thus it is likely that the right post is the phantom.

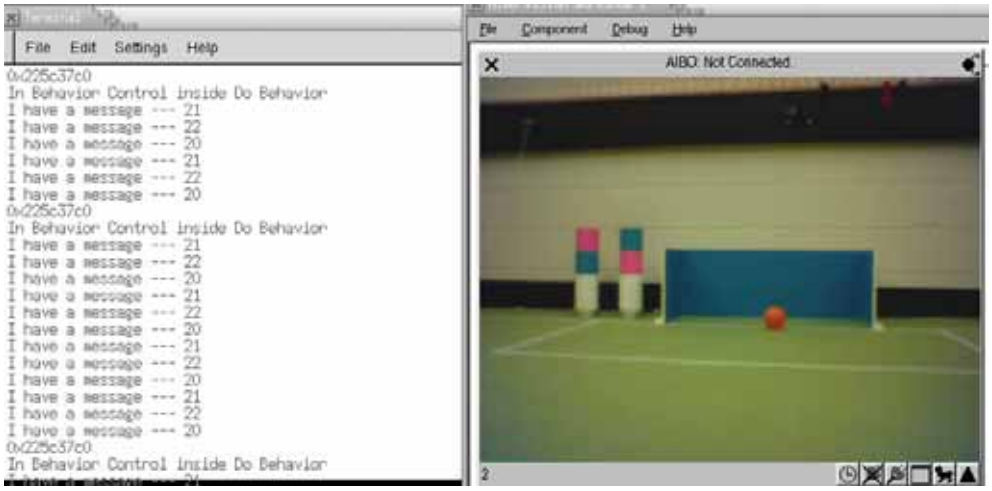
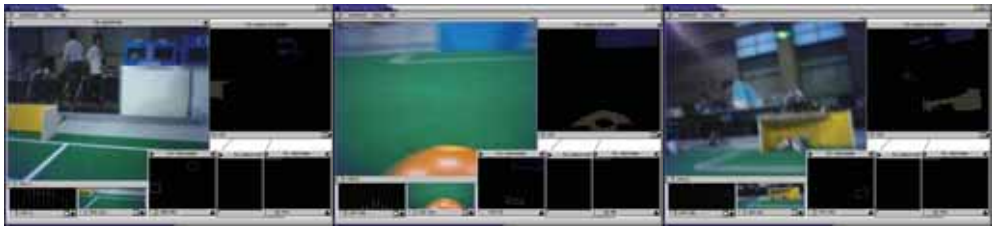


Fig. 2. The left post and the goal are in correct order, but the right post is not

The following figures show images at the RoboCup-2005 venue in Osaka where the consistency module filtered phantom objects for localization. Figure . 3 shows the processing by vision system on board the Sony AIBO. We have enlarged the captured image on board, then the blobs of color as the second largest and the objects reported by vision appear on three screens on the bottom right corner. The left most of these bottom images displays the sightings for goals.



(a) The blue timer appears as a goal with the yellow goal.

(b) Phantom goal caused by yellow pixels on the ball.

(c) A window appears as a blue goal above the real goal.

Fig. 3. Examples of competition situations where we see opposite goals in the same frame

Figure . 3 (a) shows that the blue match score and timer appear as a goal on a frame with the yellow goal. While our vision system has an analysis for filtering objects above the field of vision, the fact that the Sony AIBO has a head with three degrees of freedom and has legs that during pursuit of the ball make positions and angles of vision that cannot always rule this case out. Figure . 3 (b) shows that phantom sightings occur even with the regular color-coded objects in the field. The ball has enough yellow pixels to be confused with a yellow goal against a blue goal. Figure . 3 (c) shows another case where natural lighting and off the field objects result in phantom sightings. In this case, a window registered enough blue pixels to be reported as a blue goal on a frame that spots the yellow goal as well.

With the aid of a GUI simulator (description to follow) and the telnet connection, the models

were evaluated for all possible configurations of phantom and real sightings that involve up to three landmarks, and even in some cases 4 or 5 landmarks. Some examples of the outcomes are shown in the Figure 4. We invite the reader to attempt to decide what the reliable sightings are before exploring the results produced by the models.

The results are as follows. In Figure 4 (a) only BG and RBP are consistent. For Figure 4 (b), BG and RBP are the consistent objects while only RBP is consistent for Figure 4 (c). For these three previous cases, with Model 3 both `Cs_LBP` and `¬Cs_LBP` are -1. For Figure 4 (b) both `Cs_RP` and `¬Cs_RP` are -1. In Figure 4 (d) nothing is consistent while for Figure 4 (e) only RBP and RP are consistent.

(a) RBP BG LBP RP (b) RP RBP BG LBP (c) RP RBP LBP (d) RBP BG FG LP (e) RBP BG FG RP

Fig. 4. Interesting cases for the static models

3.4 Using A Model of Time to Improve Localization

Our previous model illustrated reasoning based only on the current reading from the sensors. It is natural that decisions on agents may be based not only on the information on the current state of the system, but also on data retrieved in the past. To illustrate how we can accomplish reasoning about the current and previous states of the environment we show a temporal expansion of the previous spatial DPL model. In a temporal model, the objects may have been visible in the previous frame or in the current frame. So we model this by considering that vision now reports sightings with respect to a time step or a frame (i.e. the predicate is now *See*(*x*,*f*)).

```
type Frame = {PF, CF}. type See(x <- Landmark, f <- Frame).
type SeeLtoR(x <- Landmark, y <- Landmark - {x}, f <- Frame).
```

Sightings may be transient (did not last across consecutive frames) or persistent (the object is in both frames).

```
type Tra(x <- Landmark).   R1: {} => ~See(x,f).   R2: {} => ~Tra(x).
R3: {See(x, PF), ~See(x, CF)} => Tra(x).
R3: {~See(x, PF), See(x, CF)} => Tra(x).  R3 > R2.
type Per(x <- Landmark).   R4: {} => ~Per(x).
R5: {See(x, PF), See(x, CF)} => Per(x).   R5 > R4.
```

Nothing is consistent unless we get at least a transient or a persistent sighting.

```
type Cs(x <- Landmark). R6: {} => ~Cs(x).
R7: {Tra(x)} => Cs(x). R7 > R6. R8: {Per(x)} => Cs(x). R8 > R6.
```

Seeing two opposite landmarks is grounds for inconsistency (even persistently or transiently).

```
R9: {Opp(x, y), Per(x), Per(y)} => ~Cs(x). R9 > R7. R9 > R8.
R10: {Opp(x, y), Tra(x), Per(y)} => ~Cs(x).
R11: {Opp(x, y), Tra(x), Tra(y)} => ~Cs(x). R10,R11 > R7.
```

What does it mean for two objects to be in a transient left-to-right order? This happens if vision sees the objects in the previous frame in that order but does not see them in the current frame in that order, or they are seen in the current frame in that order but they were not seen in the previous frame in that order.

```
type TraLtoR(x <- Landmark, y <- Landmark - {x}).
R14: {} => ~TraLtoR(x,y).
R15: {SeeLtoR(x,PF,y,PF),~SeeLtoR(x,CF,y,CF)} => TraLtoR(x, y).
R15: {~SeeLtoR(x,PF,y,PF),SeeLtoR(x,CF,y,CF)} => TraLtoR(x, y).
R15 > R14.
```

However, objects are persistently seen in a left-to-right order if the sighting of that relationship happened in the previous and current frame.

```
type PerLtoR(x <- Landmark, y <- Landmark - {x}).
R16: {} => ~PerLtoR(x,y).
R17: {SeeLtoR(x,PF,y,PF), SeeLtoR(x,CF,y,CF)} => PerLtoR(x,y).
R17 > R16.
```

Finally, seeing landmarks out of order is grounds for inconsistency. But we only overwrite those rules that may have suggested consistency.

```
R18: {LR(x, y), Per(x), Per(y), PerLtoR(y, x)} => ~Cs(x).
R18: {LR(x, y), Per(x), Per(y), PerLtoR(y, x)} => ~Cs(y).  R18 > R8.
R19: {LR(x, y), Tra(x), Per(y), TraLtoR(y, x)} => ~Cs(x).
R19: {LR(x, y), Per(x), Tra(y), TraLtoR(y, x)} => ~Cs(y).  R19 > R7.
R20: {LR(x, y), Tra(x), Tra(y), TraLtoR(y, x)} => ~Cs(x).
R20: {LR(x, y), Tra(x), Tra(y), TraLtoR(y, x)} => ~Cs(y).  R20 > R7.
```

3.5 Implementation of Temporal Models

For the temporal model, the robot executes a variant of the template method (named `Run()`). Again, the template method is executed every time a prompt from the environment demands an action and we want to do some reasoning before the action. For the localization example, arrival of a frame and its analysis by the vision module are prompts for reasoning about the sightings before sending results to the localization module. However, this variant involves other macros.

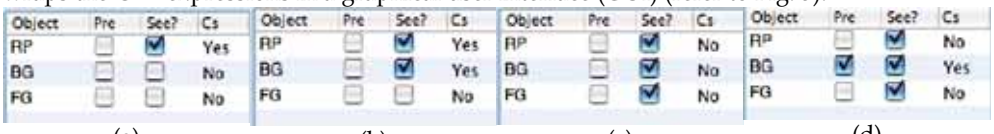
```
void  Consistency::Run()
{INIT_ALL_FALSE(); UPDATE_ALL(); CHECK_NEW_LANDMARKS();
PLACE_CS_ALL(); COPY_ALL_BOOL(); }
```

As explained earlier, the building blocks of `Run()` are procedures defined by computer generated glue code macros. The `INIT_ALL_FALSE()` macro implements the first phase. It gives definitions of C++ Boolean variables for all defined inputs and sets all Booleans

corresponding to inputs of the current frame to false. `UPDATE_ALL()` queries the reports of all the sensors in the current status of the environment, in our case obtains from the vision module reports for the current frame. We have a new intermediate phase `CHECK_NEW_LANDMARKS()` that ensures all information from sensors is labeled with its time step identifier. A sensor reading now is labeled with subindex 0, but for the previous fame the index is -1. Indexes are shifted when this macro is executed. In our case, this ensures that sightings are for the current frame and that previous sightings also have the correct fame numbers relative to the current frame. As before, `PLACE_CS_ALL()` will evaluate the output expressions using the inference engine. If a landmark is found consistent, it will forward the sighting to the localization module (or any other module that may benefit from it, like the action to kick when the front goal is visible). `PLACE_CS_ALL()` will have as many `if` statements as outputs/proofs are requested. For example, testing (evaluating) the `Cs_FG` macro, corresponds to asking the inference engine if we have a consistent sighting of the front goal. Finally `COPY_ALL_BOOL()` shifts all the current Boolean values in C++ Boolean variables to the Boolean variables corresponding to previous frames (so the previous frame values are correctly set for the next execution of the template method `Run()`). We avoid keeping the reports and evaluating predicates regarding vision reports on previous frames.

3.6 Evaluation of the Temporal Model

The entire architecture, and the models, were also validated in a Graphical User Interface (GUI) simulator. That is, the `roboconsistency.h` files produced for a model as output by DPL can be used by the GUI simulator as well. A user interacting with the GUI simulator can set up the vision reports (for example, set up a scenario where the front goal and front post are visible and the front post is to the right of the front goal). The GUI simulator then indicates which of these landmark sightings are regarded as worth forwarding to the localization module. This simulator facilitates the debugging of the entire architecture without having to execute the consistency module on the robot. This is particularly useful in the evaluation of temporal properties. On the robot, validation is particularly hard because sighting errors only occur sporadically at a frame rate higher than 25 Hz. Our simulator allows reproducible scenarios of what vision might report to the localization system and this was used to validate the correctness of the PL expressions resulting from a model. The inference engine is also the same in the simulator and the on-board execution module on the robot. In order to provide a way to consistently set and evaluate a scene, the simulator wraps the C++ expressions in a graphical user interface (GUI) (refer to Fig. 5).



(a) (b) (c) (d)

Fig. 5. (a) The simulator showing one particular object (RP) as visible in the current frame. (b) Two consistent objects in the current frame. (c) Three inconsistent sightings in the current frame. (d) The persistent back goal (BG) wins over the temporary sightings of the right post (RP) and the front goal (FG)

The user of the simulator places landmarks on rows, and the succession of rows represents

the order in which these objects are seen (with top to bottom representing left to right in the field of vision). The first column shows the landmark. The next two columns allow the user to select what the visibility state is for the previous (`Pre`) and current (`See?`) frames. The rightmost column shows the output of the consistency module (`Cs`) after performing its reasoning. Furthermore, the GUI allows the dragging and dropping of objects to change the order, as well as addition (`Add`) and deletion (`Delete`) of landmarks.

Fig. 5 (b) shows two consistent sightings. Even though the two landmarks RP and BG were not visible in the previous frame, they are consistent with each other, allowing them to be forwarded on to the localization module. In fact, this is the same result that a traditional value domain reasoning system would obtain. This is also true for Fig. 5 (c) where we can see three objects that are inconsistent with each other. Since all the objects only occur within one single frame, the only conclusion that can be drawn is that nothing is consistent in that scene. Once information varies over time, a richer belief about the environment can be formed. Fig. 5 (d) shows the same scenario as Fig. 5 (c), but this time the temporal properties of the visible objects vary. The back goal that was visible in the previous frame as well as the current frame is given precedence over the right post and the front goal that were only visible in a single frame.

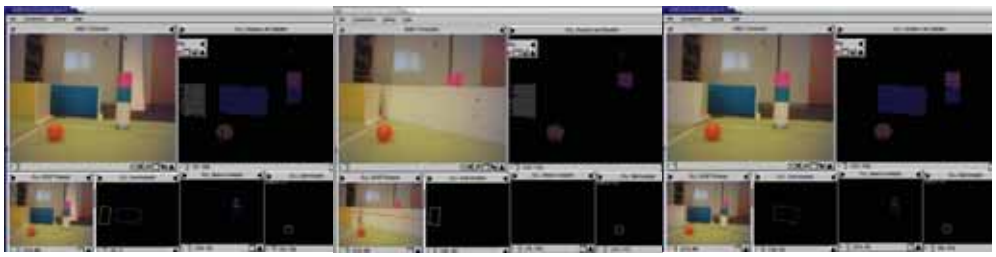
Temporal and spatial tests can be combined as in Fig. 6 (a). Objects that are consistent in either space or both space and time are ruled as being consistent in the world view of the system. Only inconsistencies that persist over both space and time will force the system to conclude that nothing is consistent (Fig. 6 (b)).

Fig. 6. Whether an `Object` is visible is indicated by `See?` for the current frame and by `Pre` for the previous frame. Column `Cs` shows whether the corresponding object is concluded to be consistent. (a) The two persistent landmarks (RP and BG) are consistent with each other, but inconsistent with the front goal (FG). (b) Nothing is consistent in this view

3.6.1 Evaluation on the Robot

We also have analyzed the effectiveness of the approach in a Sony AIBO ERS-7 in the lab. Fig. 7 shows a lab setting where we can rapidly produce opposite goals in a frame and immediately after block one goal, or the other. In the log, we found sequences where the robot is seeing only the front goal and reports it as consistent. When the back goal appears as well, for that first frame, the front goal remains consistent and the back goal is labeled inconsistent (note that in the discussion of the KF, MM, and MCL localizers we indicated that the frame with both goals becomes not usable). If the back goal persists with the front goal for one more frame, then both goals are now labeled inconsistent (note that the model

can easily be adjusted if a different effect is desired besides having two consecutive frames with both goals to rule them out). If the front goal drops out, then the back goal in the previous frame and the back goal in the current frame both become consistent. We have no space here to discuss more examples of the versatility of the modeling here, but using the RP also helps since the blue goal is in the right left-to-right order with respect to the post.



(a) Both goals are visible. (b) The blue goal is covered. (c) The yellow goal is covered.

Fig. 7. Lab Examples. A setting that allows sequences of frames where two goals are reported by vision. Goals can be covered and uncovered quickly

4. Plausible Logic for the Referee

One of the most crucial aspects of soccer is the offside rule (rugby, hockey and many others have similar variants). Typically, the rule is a source of much debate, and almost every soccer fan forgets one or more of the exceptions when first presenting the rule to a novice. The rule represents an interesting resource for the defense, and its comprehension is crucial for the attacking team. Naturally, soccer referees (main and assistant) are mostly judged by their ability to officiate the rule.

If robotic players are to participate in a competition in 2050 and defeat the human world champion they would need to reason about these types of rules. Moreover, we can foresee that an artificial agent can enforce the rules in leagues like the simulation league or even the Sony AIBO competition as there are now video analyzers that recognize players, landmarks and the ball (Ruiz-delSolar *et al.*, 2006).

The official rules of the game, as per the FIFA web site indicate that Rule 11 corresponds to the off-side rule.

> *It is not an offence to be in an offside position. A player is in an offside position if he is*
>
> - *nearer to his opponents' goal line than both the ball and second last opponent*
>
> *A player is not in an offside position if he is*
>
> - *on his own half of the field of play or*
> - *level with the second last opponent or*
> - *level with the last two opponents*
>
> *A player in an offside position is only penalized if, at the moment the ball touches or is played by one of his team, he is, in the opinion of the referee*
>
> - *interfering with play or*
> - *interfering with an opponent or*

- *gaining an advantage by being in that position.*

There is no offside offence if a player receives the ball directly from

- *a goal kick or*
- *a throw-in or*
- *a corner kick.*

We now describe this with a model using PL with the programming language DPL. We will mimic closely the official wording, although simpler equivalent models can be described more succinctly. We define the objects we need to talk about as the last two opponents and the ball.

```
type Last2Opponents = {Opp1, Opp2}.
type Objects = {Ball} + Last2Opponents.
```

We also need to describe the possible types of activities for a player and the possible ball transfers.

```
type Plays = {IfPlay, IfOpp, TkAd}.
type Transfers = {GlKk, ThwIn, CnrKk}.
```

The robotic referee would need to deduce from its sensors whether a player is level with the second last opponent.

```
type lvl(x <- Last2Opponents).
```

Also, using its sensors, it must identify the involvement of the player in play and the types of transfers that result in no offside.

```
type play(x <- Plays).
type xfer(x <- Transfers).
```

The default situation is that a player is not committing an offense, as per the official rules above. Note that we are naming rules with more meaningful names that just a rule identifier.

```
NoOffence: {} => ~offsideOffence.
```

The remaining rules are now clear from the similarity with the FIFA's Rule 11.

```
Offence: {offsidePosition, active} => offsideOffence.
Offence > NoOffence.

NotOffside: {} => ~offsidePosition.

Offside: {nearr(x) | x <- Objects} => offsidePosition.
Offside > NotOffside.

OwnHalf: {ownHalf} => ~offsidePosition.  OwnHalf > Offside.

lvl: {lvl(x)} => ~offsidePosition.  lvl > Offside.
```

```
NotActive: {} => ~active.

Active: {play(x)} => active.  Active > NotActive.

Transfer: {xfer(x)} => ~offsideOffence.  Transfer > Offence.
```

5. Plausible Logic for the RoboCup@Home

RoboCup@Home is a league that concentrates on real-world applications for robotics. It has a strong focus on interaction between autonomous robots and humans, aiming at the development of applications that can assist humans in everyday life (van der Zant & Wisspeintner). We believe that such applications will become prevalent in the near future and will be an integral part of our lives in areas such as public transport, housework, care, and medicine. We have explored human-robot interfaces for assisting learning of blind children (Bartlett *et al.*, 2003) and e-mail between blind adults (Estivill-Castro & S., 2006). As originally proposed (van der Zant & Wisspeintner), RoboCup@Home anticipates many applications of robots assisting elderly humans with situations like emergencies.

We have presented, as part of the Open Challenge in RoboCup 2007, the use of rules in DPL for a scenario where an elderly lady (Grandma) lives alone at home. While she does not require constant care, raising an alarm in an emergency or if something unexpected happens can be of vital importance. We now illustrate the constructing of these rules for our scenario. We use vision as the main source of input and information about the environment since RoboCup@Home settings can hardly have specific sensors for robots. Again, the suggested methodology for building a model is to introduce the rules incrementally, adding consideration of possibly visible objects one at a time.

We presume that the raising of an alarm could be forwarded to a behavior module on the robot. Thus, *alarm* means an alarm should be raised about Grandma's welfare. By default, no alarm should be raised. *Default*: {}⇒~*alarm* The frames analyzed by vision have attached a frame identifier *f*. $F^{t',t}$ denotes the sequence of frames between two points in time, *t* and *t'*. The variable *now* denotes the point in time when a decision for an alarm is being made. For a short time in the past, say half an hour, we use *short*, but *long* is a long time, say 8 hours. The variables *long* and *short* could represent a collection of tunable parameters, for those rules that use them. Again, sightings from the sensors are plausible assumptions and *See*(*x*,*f*) is true if the vision module reports that object *x* is visible in frame *f*. Sorry Grandma, but to your robotic helper you are just an object.

A prolonged absence of *Grandma* is reason to raise an alarm. PL represents this as a definition for what *absence* means (*absence* = $\forall f \in F_{now}^{long}$ ~*See*(*Grandma*,*f*)), one plausible rule *Absence*: {*absence*}⇒*alarm*, and an instance of the priority relation *Absence* > *Default*. In DPL this can be expressed as follows.

```
Default: {} => ~alarm.
Absence: {absence} => alarm.  Absence > Default.
```

Grandma is likely to be in trouble (suffered a fall) if not standing, that is *Horizontal*. If Grandma is horizontal, then Grandma is by necessity visible. Our implementation can

assume that the analysis of a frame by vision can not assert *Horizontal*(*Grandma*,*f*) without also asserting *See*(*Grandma*,*f*) over matching frame ranges. A fall is likely if Grandma is *Horizontal* for a short time. Thus, formally, we need again a definition (*lying* = $\forall f \in F_{now}^{short}$ *Horizontal*(*Grandma*,*f*)), two refining rules (*Lying*: {*lying*} ⇒ *fall*, and *Fall*: {*fall*} ⇒ *alarm*), and one more instance in the priority relation *Fall*>*Default*. This can be expressed by the following DPL rules.

```
Lying: {lying} => fall.

Fall: {fall} => alarm.  Fall > Default.
```

With the simple model so far, we can already handle two cases that might be cause for concern and raise an alarm. However, in a real-world scenario, it may be perfectly valid for Grandma to lie down and rest. It is likely that when Grandma is lying down on her bed, that she may be resting. We now model this refinement. We assume that the robot can sense *Over*(*x*,*y*,*f*), that means that object *x* is over object *y* in frame *f*. *Over*(*x*,*y*,*f*) implies *See*(*x*,*f*) and *See*(*y*,*f*). There is no possibility that Grandma can be absent and over her bed, so rules *Absence* and *OnBed* can never conflict. Mathematically, *onBed* = $\exists f \in F_{now}^{short}$ *Over*(*Grandma*,*Bed*,*f*). The refinement is introduced with a rule that indicates the exception *OnBed*: {*onBed*} ⇒ ~*alarm*. And the priority over the rule that raises an alarm when lying *OnBed* > *Lying*. The corresponding DPL rules are as follows.

```
OnBed: {onBed} => ~fall.  OnBed > Lying.
```

However, if Grandma stays in bed for too long and is not getting up, this may still be cause for concern. Therefore, if Grandma is horizontal for a long time, raise an alarm regardless of where she is. We specify what it is to be lying for too long as *LyingLong*: {*lyingLong*} ⇒ *notGettingUp*, where *lyingLong* = $\forall f \in F_{now}^{long}$ *Horizontal*(*Grandma*,*f*). Then, we have an exception to the exception with the rule *NotGettingUp*: {*notGettingUp*} ⇒ *alarm* and another instance of the priority relation *NotGettingUp* > *Default*. So we can extend our DPL model once more to take the amount of time that Grandma has been lying down for into account.

```
LyingLong: {lyingLong} => notGettingUp.
NotGettingUp: notGettingUp => alarm.  NotGettingUp > Default.
```

Grandma is likely to be okay if the long time she is lying is at night and on the bed. At night you cannot see the sun. Grandma is not likely to get up at dawn, and may go to bed before dark. All we know is that bedtime overlaps nighttime. This leads to the following definition *nighttime* = $\exists f \in F_{now}^{long}$ ~*See*(*Sun*,*f*). We model the new revisions by another rule *Nighttime*: {*nighttime*} ⇒ ~*notGettingUp*. This is a revision on when it is OK to be lying too long *Nighttime* > *LyingLong*. The following DPL code handles this scenario.

```
Nighttime: {nighttime} => ~notGettingUp.  Nighttime > LyingLong.
```

To continue with this illustration, we ask the question what happens if Grandma is not home alone. Perhaps we want to raise an alarm if a stranger is looming over Grandma. For purposes of this scenario, a stranger is anyone other than Grandma. The presence of a stranger is not alarming unless Grandma is horizontal. We define the condition our sensors

can detect as *looming* = $\exists f \in F_{now}^{short}$ *See*(*Stranger*,*f*) $\wedge$ $\forall f \in F_{now}^{short}$ *Horizontal*(*Grandma*,*f*). Then, we define the rule *Looming*: {*looming*}$\Rightarrow$*alarm*. And place it in the hierarchy with *Looming*>*Default*. This can be expressed in DPL as follows.

```
Looming: {looming} => alarm.  Looming > Default.
```

To better illustrate how these rules work, Figure 8 summarizes the relationship between the rules shown above. To cope with more complex scenarios, additional rules can be created and priority relations can be added to resolve potential conflicts between the existing set of rules and the newly added set.

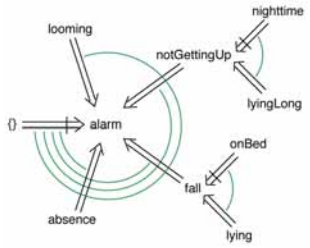


Fig. 8. Grandma's rules represented graphically. A priority is represented by an arc. The anticlockwise end beats the clockwise end

5.1 Implementation and Evaluation

The DPL model for Grandma's helper was implemented and evaluated in a similar fashion to the robotic soccer models introduced earlier. Using the GUI simulator, all combinations of inputs were tested to verify that the response (*alarm* or no *alarm*) was consistent with the model's discussion. An instance of the simulator using the complete set of rules is shown in Figure 9.

The user can set the conditions as perceived by a virtual vision module. If the user changes the inputs, the engine attempts to prove the *alarm* output. The "Proof" output column for *alarm* is set to +1 if an alarm should be raised and to -1 if no alarm should be raised. An output of 0 would indicate that the current situation cannot be decided either way (because a proof always leads to loop). This cannot (and did not) occur with the given set of rules. The "Negation" output column shows the results of proofs of ~*alarm*.

Fig. 9. Evaluation of the Grandma-Alarm model in the Plausible Test GUI simulator. The state of the DPL inputs is set on the left side of the screen while the output (the proof result from the engine whether *alarm* should be set or not) is shown on the right hand side

The same model and the DPL proof engine were then used on an ERS-7 Sony AIBO for the RoboCup@Home open challenge. We used the same vision module of our robotic soccer code. Any person dressed in blue is easily recognized using the code that recognizes the blue goal, while we set the bed yellow so we reused the modules for identifying the yellow goal. An orange circle passes for the sun, and thus the code for the module to recognize balls was reused. Any other object, when visible, was perceived as a stranger.

While most of the robotic software used for the league (such as vision, the behavior module, and the networking code) could be re-used some changes needed to be made to accommodate the Grandma scenario. A new module is introduced with the template method and the phases we discussed before. The template method executes every time a frame is ready and analyzed by vision. This new module forwards alarm signals to other modules (a behavior module or a network connection if an *alarm* takes the value true for the current frame). Naturally, some of the definitions must be translated into concrete detectable sensor information. Thus, some C++ code needs to be produced. For example, in a soccer game, the goal would never be expected to be in a vertical position. Since it does make a difference whether Grandma is standing upright or is lying in a horizontal position, a method was added that would determine whether the dimensions of the blue object (if reported as visible by the vision module) were horizontal (wider than its high) or vertical (higher than its width). Similarly, if both goals (i.e., both Grandma and her bed) were visible, C++ code needs to be added to compare their relative positions to provide the information whether Grandma is lying on the bed or not. C++ code needs to be added for the other terms defined in PL that demand information from processing sensory input. To determine if Grandma had been lying down for long, we developed new C++ code that counts the number of frames that Grandma had been seen in a horizontal position.

6. Conclusion

One of the most satisfying aspects of our implementation is that it has proven efficient enough to be running on board a Sony AIBO while in competition in the soccer 4-legged league or in RoboCup@Home. Table 1 shows the CPU-timings on board an ERS-7 running our C++ implementation with three different models and two situations. It may be surprising that while the robot was in the playing state, chasing the ball and executing kicks, the execution is faster than while standing as a goalie. However, the standing situations have usually on average 2 landmarks per frame. But while playing, frames with 2 objects in sight are less frequent. In all cases, the inference engine is executed six times per frame, to verify if each of the landmarks is consistent.

Model	Activity	Phase 1 and Phase 2	95 % Confidence Interval	Phase 1, Phase 2, and Phase 3	95 % Confidence Interval	Net Phase 3
4	Chasing	749 µs	± 7 µs	931 µs	± 8 µs	182 µs
4	Standing	1,438 µs	± 31 µs	1,687 µs	± 35 µs	249 µs
3	Standing	407 µs	± 15 µs	622 µs	± 17 µs	215 µs
2	Standing	209 µs	± 13 µs	371 µs	± 17 µs	162 µs

Table 1. CPU-times for the 3 phases of our template method on a Sony AIBO ERS-7

7. References

Antoniou, G. (1997). *Nonmonotonic Reasoning*. MIT Press, Cambridge, Mass. ISBN 0-262-01157-3.

Bartlett, B.; Estivill-Castro, V.; S., Seymon & Tourky, A. (2003). Robots for pre-orientation and interaction of toddlers and preschoolers who are blind. In Roberts, J. & Wyeth, G., editors, *Proceedings of the 2003 Australasian Conference on Robotics and Automation*, Brisbane, Australian Robotics and Automation Association Inc, Queensland Centre for Advanced Technologies (QCAT). CD-ROM (paper 13.pdf) ISBN 0-9587583-5-2.

Billington, D. & Rock, A. (2001). Propositional plausible logic: Introduction and implementation. *Studia Logica*, 67:243–269. ISSN 1572-8730.

Billington, D.; Estivill-Castro, V.; Hexel, R. & Rock, A. (2005). Non-monotonic reasoning for localisation in robocup. In Sammut, C., editor, *Australasian Conference on Robotics and Automation*, Sydney, Australian Robotics and Automation Association. ISBN 0-9587583-7-9.

Billington, D.; Estivill-Castro, V.; Hexel, R. & Rock, A. (2006). Using temporal consistency to improve robot localisation. In Lakemeyer, G. & Sklar, E., editors, *International RoboCup Symposium*, Bremen, Germany, Springer-Verlag Lecture Notes in Computer Science. Volume 4434, pages 232-244, 2007. ISBN 978-3-540-74023-0.

Billington, D. (2005). The proof algorithms of plausible logic form a hierarchy. In Zhang, S. & Jarvis, R., editors, *Proceedings of the 18th Australian Joint Conference on Artificial Intelligence*, volume 3809, pages 796–799, Sydney, Australia, Springer Verlag Lecture Notes in Artificial Intelligence. ISBN 3-540-30462-2.

Brooks, R.A. (1991). Intelligence without reason. In Myopoulos, R. & Reiter, R., editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 569–595, San Mateo, CA, ICJAI-91, Morgan Kaufmann Publishers. Sydney, Australia. ISBN 1-55860-160-0.

Compton, P.J. & Jansen, R. (1990). A philosophical basis for knowledge acquisition. *Knowledge Acquisition*, 2(3):241–257. ISSN 0001-2998.

Estivill-Castro, V. & S., Seymon. (2006). Mobile robots for an e-mail interface for people who are blind. In Lakemeyer, G. & Sklar, E., editors, *International RoboCup Symposium*, Bremen, Germany, Springer-Verlag Lecture Notes in Computer Science. Volume 4434, pages 338-346, 2007. ISBN 978-3-540-74023-0".

Fox, D.; Burgard, W. & Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligent Research*, 11:391–427. ISSN 1076-9757.

Gutmann, J.-S. & Fox, D. (2002). An experimental comparison of localization methods continued. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, v 1, pages 454–459, Lausanne, Switzerland, IEEE. ISBN 0-7803-7398-7.

Haemmi, R. & Hartmann, S. (2006). Modeling partially reliable information sources: A general approach based on Dempster-Shafer theory. *Information Fusion*, 7:361–379. ISSN 1566-2535.

Kahney, L. (2003). The new pet craze: Robovacs. Wired Magazine. June, 16th; visited Septenber 10th, 2003, www.wired.com/news/technology/0,1282,59249,00.html.

Marek, V.W. & Truszczynski, M. (1993). *Nonmonotonic Logic: Context-Dependent Reasoning*.

Springer Verlag, Berlin. ISBN 0387564489.

Rich, E. & Knight, K. (1990). *Artificial Intelligence*. McGraw-Hill Higher Education, NY, second edition. ISBN 0070522634.

Rock, Andrew. The DPL (decisive Plausible Logic) tool. Technical report, (continually) in preparation, available at www.cit.gu.edu.au/ ,˜arock/.

Rock, A. & Billington, D. (2000). An implementation of propositional plausible logic. In Edwards, J., editor, *23rd Australasian Computer Science Conference*, volume 22(1) of *Australian Computer Science Communications*, pages 204–210, Canberra, IEEE Computer Society, Los Alamitos. ISBN 076950518X.

Ruiz-delSolar, J.; Loncomilla, P. & Vallejos, P. (2006). An automated refereeing and analysis tool for the four-legged league. In Lakemeyer, G. & Sklar, E., editors, *International RoboCup Symposium*, Bremen, Germany, Springer-Verlag Lecture Notes in Computer Science. in press.

Russell, S. & Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Englewood Cliffs, NJ, second edition. ISBN 0130803022.

Thrun, S.; Bennewitz, M.; Burgard, W.; Cremers, A.B.; Dellaert, F.; Fox, D.; Hahnel, D.; Rosenberg, C.R.; Roy, N.; Schulte, J. & Schulz, D. (1999). MINERVA: A tour-guide robot that learns. In *KI - Kunstliche Intelligenz, 23rd Annual German Conference on Artificial Intelligence*, volume 1701, pages 14–26. Springer Verlag Lecture Notes in Computer Science. ISBN 3-540-66495-5.

Thrun, S.; Fox, D.; Burgard, W. & Dellaert, F. (2001). Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128:99–141. ISSN 0004-3702.

van der Zant, T. & Wisspeintner, T. Robocup X; a proposal for a new league where robocup goes real world. www.robocupathome.org.

Veloso, M.; Uther, W.; Fujita, M.; Asada, M. & Kitano, H. (1998). Playing soccer with legged robots. In *Proceedings of IEEE/RSJ Intelligent Robots and Systems Conference (IROS-98)*, volume 1, pages 437–442, Victoria, Canada. ISBN 0-7803-4465-0.

Wallich, P. (2001). Mindstorms - not just a kids toy. *IEEE Spectrum*, pages 53–57. ISSN 0018-9235.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. John Wiley & Sons, NY, USA. ISBN 047149691X.