Non-monotonic Reasoning for Localisation in RoboCup

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

Abstract
Initial progress in symbolic reasoning was mostly discarded by the autonomous/mobile robotics community because robots usually face unpredictable environments and incomplete knowledge. As an alternative, reactive systems have taken over, heavily influenced by the works of Brooks [Brooks, 1991] and others. However, we demonstrate here an application of non-monotonic logic for the solution of localisation problems. While Plausible Logic [Billington and Rock, 2001; Rock and Billington, 2000] is implementable, we show here it is operable in real time on a Sony AIBO in the RoboCup (robotic soccer) context.

1 Introduction
Robots are today a reality. Moreover, robots have moved from assembly lines to being around human beings. 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 and Truszczynski, 1993; Rich and Knight, 1991] (this is also illustrated by the 4 chapters dedicated to uncertain knowledge and reasoning in [Russell and 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 paper will focus on dynamically selecting proper inputs for localisation. Localisation 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 localising, with a loose analogy to when people use previous knowledge to explore a city 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 localisation (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) localisers). We do not advocate their elimination. In fact, we use the Monte Carlo Localisation approach for localising Sony AIBO robots in the 4-legged league of RoboCup. However, we introduce non-monotonic reasoning as a filter before the localisation process takes observations as inputs.
We propose to use non-monotonic reasoning to accept the inconsistent information from sensors 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 to make a decision and then act. We also want some assessment of the likelihood of that state for the decision making process.

We demonstrate our initial system with a consistency module used by the Mi-Pal team in RoboCup 2005 in Osaka, Japan (www.robocup.org). The team participated in the legged league [Veloso et al., 1998]. The current robotic platform is the Sony AIBO robot. This presents a dynamic environment for multiple legged robots. Robotic soccer is the most challenging endeavour from the perspective of multi-agent technology [Wooldridge, 2002]. It consists of an environment that is non-deterministic, inaccessible, continuous, dynamic, and that has teams and adversaries. This makes the robocup environment an ideal context for a reactive architecture. While most participants at RoboCup use hybrid architectures, we believe this is the first application of non-monotonic reasoning in such a dynamic, non deterministic environment.

2 State of the Art

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 localisation, alternative techniques have emerged. Most notably, grid-based Markov localisation and Monte Carlo localisation [Fox et al., 1999; Gutmann and 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. 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. Namely, 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 this is not possible. 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. The localization approaches need to estimate Prob(visible scene|p̂), where the visible scene is formed of a description of all visible objects, and where p̂ is a vector for 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|p̂) and Prob(See_back_goal|p̂) (for example when seeing both goals). The problem with this is that because p̂ 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 ∩ See_back_goal|p̂) = 0, for all postures p̂. However, we just indicated that representing Prob(See_front_goal ∩ See_back_goal|p̂) as a function of Prob(See_front_goal|p̂) and Prob(See_back_goal|p̂) 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. For example, if we see both goals in one frame, the safest option is to rule out both observations and not to use either as input in localisation to update the current belief. In fact, the system falls essentially into the realm of classical logic that applies rules of thumb as sanity checks to the readings from sensors.

Our proposal is to use Plausible Logic (PL) [Billington and Rock, 2001; Rock and Billington, 2000], a formalism for non-monotonic reasoning, to perform sophisticated analysis of the objects reported by a vision module before using these observations in localisation modules (which can be constructed based on Kalman filters, Markov models, or under the Monte Carlo approach).

3 Plausible Logic (PL)

Non-monotonic reasoning is the capacity to make inferences from a database of beliefs and to correct those as new information arrives that make previous inferences
invalid. Although several non-monotonic formalisms have been proposed [Antoniou, 1997], PL is 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 by a variety of algorithms; each providing a certain degree of trust in the conclusion. If only factual information is used, PL essentially becomes classical propositional logic. However, when determining the provability \(^1\) 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 the formulas that have been proved. It assigns the value 0 when the formula cannot be proved and attempting so will cause infinite recursive looping. It assigns the value −1 when the formula is not provable and does not generate a loop.

In PL all information is represented by three kinds of rules and a priority relation between those rules. The first type are strict rules that use the strict arrow \(\rightarrow\) and we use them to model facts which 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 \(\text{human}(x) \rightarrow \text{mammal}(x)\).

Plausible rules \(A \Rightarrow l\) use the plausible arrow \(\Rightarrow\) 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 \(\text{bird}(x) \Rightarrow \neg \text{fly}(x)\). 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 \(A \rightarrow \neg l\) 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 \(\{\text{sick}(x), \text{bird}(x)\} \rightarrow \neg \text{fly}(x)\). Defeater rules prevent conclusions which would otherwise be too risky. This happens 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 paper we actually demonstrate the expressive power of this aspect of the formalism. For example from

\[
\begin{align*}
\{\} & \rightarrow \text{quail}(Quin) & \text{Quin is a quail} \\
\text{quail}(x) & \rightarrow \text{bird}(x) & \text{Quails are birds} \\
R_1: \text{bird}(x) & \Rightarrow \text{fly}(x) & \text{Birds usually fly} \\
\end{align*}
\]

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

\[
\begin{align*}
\{\} & \rightarrow \text{quail}(Quin) & \text{Quin is a quail} \\
\text{quail}(x) & \rightarrow \text{bird}(x) & \text{Quails are birds} \\
R_2: \text{quail}(x) & \Rightarrow \neg \text{fly}(x) & \text{Quails usually do not fly} \\
\end{align*}
\]

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 perhaps can say that \(R_2\) is more informative as it is more specific and so we add \(R_2 > R_1\) to a knowledge base representing the beliefs of a robot who 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 3 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. The models presented here are each a progressive refinement of the previous one.

4 Modelling with Logic

Self-localisation determines at time \(t\) the probability function \(\text{Prob}(\vec{x}_t)\) of being currently at a pose described by the vector \(\vec{x}_t\). The inputs to the localisation algorithms are observations \(\vec{a}_t\) and executed actions \(\vec{a}_t\). A generic framework for KF, MM and MCL [Gutmann and Fox, 2002; Thrun et al., 2001] indicates that the robot will iteratively use a motion model to make a prediction for \(\vec{x}_{t+1}'\) using \(\text{Prob}(\vec{x}_t)\) and \(\vec{a}_t\). Then, it will correct the prediction using the data \(\vec{a}_t\) from its sensors to correct \(\vec{x}_{t+1}'\) and produce the new \(\text{Prob}(\vec{x}_{t+1})\).

MCL and its variants have been shown to be superior to the Extended Kalman Filter [Gutmann and Fox, 2002] and to vanilla MM [Gutmann and Fox, 2002]. They seem to be the most effective method for the 4-legged league [Röfer and Jüngel, 2004]. They also are known as particle filters and have parallels with non-parametric statistics in that they are universal density approximations (no need to impose a parametric model on the distribution of \(\text{Prob}(\vec{x}_t)\)). They can accommodate arbitrary sensors, motion models and noise properties. They naturally focus their computational effort in those areas believed to be more relevant, and controlling the number of particles allows regulation of computational effort. Moreover, they are significantly simpler to implement.

The MCL approach still has some disadvantages. The stochastic nature of the algorithms implies that the number of particles can not be small. Vanilla MCL does not handle the kidnap problem well, and thus it needs to model the possibility of a kidnap by randomly introducing new particles in the space. It also seems slow to converge if the sensors are too accurate and until recently little theoretical foundation existed for some of its fixes [Thrun et al., 2001]. A derivation from the theory of Bayes filters [Thrun et al., 2001] shows that the vanilla
MCL (under some general assumptions\(^2\)) estimates the belief of the pose recursively.

Because of these drawbacks, it is very 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 this is best illustrated by the use of domain knowledge to filter some observations out. As we already discussed, we know that the Sony AIBO cannot see both goals when in the soccer field. Therefore, most teams participating in the competition perform a sensible sanity check; that is, they will filter out observations from the vision system that indicate that opposite goals were seen in the same frame. These inconsistent \(^3\) inputs are simply, partially or not at all, used for localisation.

The situation becomes difficult to manage, as entangled with the localisation code is a series of logical tests which 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 localisation module.

### 4.1 Modelling without Plausible Logic

In this section we show that modelling the field of RoboCup 2005\(^4\) for the 4-legged league without PL results in a large model, where little can be inferred. This illustrates that whatever alternative method is used, it will 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.

Each 2005 4-legged league soccer field has fundamentally 6 landmarks for localisation. These are two goals (one yellow and the other blue) and four posts. Each post has two colours, and pink is always one of these. The two post near the blue goal have blue as one of the colours while the two post on the yellow side have yellow. This colour 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). The domain knowledge that these 6 objects exist and are arranged in the field as just described can be modelled in many ways; however, we show here the most succinct way we have found without PL (more than 3 models without PL were constructed).

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\(^\circ\), but to simultaneously see LP and RBP would require a view greater than 67.5\(^\circ\). Because there are 6 landmarks, there are 6 \(\times 5 = 30\) ordered pairs (ways of observing 2 landmarks) and 6 \(\times 5 \times 4 = 120\) ordered triples (ways of observing 3 distinct landmarks). This results in 30 literal axioms of the form \([-\text{Opp}(x, y)]\) (to indicate for example that both goals are opposites, and that a goal and a post are not opposites) plus 120 axioms for the observation of triplets. All these can be simplified to 6 atomic axioms and 21 clausal axiom schemas (a preprocessor would convert those 21 clausal axiom schemas into atomic axioms).

The challenge is then to consider what the always imperfect vision module may report. That is, in a single frame, the analysis of an image may actually perceive two blobs of yellow colour 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. And what we are arguing here is that if all those ways of ruling out sightings of landmarks are not concentrated in a single module represented in logic, then the software is very likely to have such rules in several modules, resulting in high coupling of several modules, and more seriously, in inconsistent modelling of the reasons why some sightings are ruled out before they are used for localisation.

As the robots move to more realistic environments more reasoning is needed (now the new field does not have a barrier of white around it, there are more phantom objects in the audience \(^5\)). Vision may report 0, 1, 2, 3, 4, or even 6 landmarks in a single frame (because of domain knowledge we know that if 6 landmarks are reported there must be at least 3 phantoms; and this is the type of reasoning we are formalising here). If vision does not report a landmark, or only reports seeing one landmark; then this is an easy case, since for only one frame we have no other information to rule this out. We

\(^1\)The RoboCup-05 setting had blue score markers that were identified as blue goals behind yellow goals.
can formalise this as follows.
\[
\{\text{See}(x)\} \cup \{\neg \text{See}(y) : y \in \text{Landmarks} - \{x\}\} \rightarrow \text{Cs}(x).
\]

That is, if we do not see any landmark, no landmark sighting is forwarded for localisation. If we only see one landmark, then that landmark is regarded as consistent and forwarded for localisation (here \text{See}(x) means the vision module has reported object \(x\) in the frame and \text{Cs}(x) means the consistency filter believes \(x\) should be an input for localisation).

Things get more complicated as we move to frames where we identify exactly two objects. We formalise this by \(C(x, y) = \{\text{See}(x), \text{See}(y)\} \cup \{\neg \text{See}(z) : z \in \text{Landmarks} - \{x, y\}\}\). There are several subcases for this case. In particular, \(x\) and \(y\) may be adjacent landmarks as the robot rotates from the centre of the soccer field (for example, LP and FG). In this case, we may have the following rules
\[
\begin{align*}
\{\text{SeeLtoR}(x, y), \text{FactLtoR}(x, y, z)\} \\
\cup C(x, y) & \rightarrow \text{Cs}(x, y) \quad (1) \\
\{\text{SeeLtoR}(y, x), \text{FactLtoR}(x, y, z)\} \\
\cup C(x, y) & \rightarrow \text{Cs}(1, x, y) \quad (2) \\
\{\text{Cs}_1(x, y), \text{Post}(x), \text{Goal}(y)\} & \rightarrow \text{Cs}(x) \quad (3) \\
\{\text{Cs}_1(x, y), \text{Post}(x), \text{Post}(y), \text{BigSmall}(x, y)\} & \rightarrow \text{Cs}(x). \quad (4)
\end{align*}
\]

That is, if vision reports exactly two objects \(x\) and \(y\), and in that frame vision observes \(x\) to the left of \(y\) (\text{SeeLtoR}(x, y)), when we know that these objects appear adjacent in the order \(x, y\) and some third object \(z\) (from our domain knowledge), then we can forward the objects to localisation, refer to Rule (1). However, if we see them in the reverse left to right order with respect to their position in the domain, we take it that one of them is a phantom. The predicate \(\text{Cs}_1(x, y)\) means only one of \(x\) and \(y\) is consistent with the domain knowledge in Rules (2)-(4). For the purposes of this paper we have added two simple rules to discriminate when one of two object sightings is inconsistent. Rule (3) simply says that when the only 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 largest of two inconsistent posts is to be forwarded as input for localisation since it is harder to perceive a large phantom post.

These four rules succinctly represent a setting in logic whose reasoning would otherwise be embedded in highly coupled C++ code, but more importantly, with the analysis of other cases, the number of rules needed grows rapidly. We have no space here to discuss what are the rules for when vision reports two objects which are landmarks that have a third landmark in between them (for example, we see the two front posts but not the front goal because the goalie blocks it). And again, we need to analyse if vision reports them in the expected left to right order or not. Then, there is the case of two landmarks exactly with two missing landmarks in between (an example of this case is when vision reports both goals). Complete rules for all these cases can be formulated (a total of 6 more rules).

Then, we have to progress to when vision reports exactly 3 landmarks. All subcases here result in 26 more rules (that we omit for lack of space). Seeing exactly 4 landmarks results in 120 rules and so on. We see that this analysis rapidly results 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 colour). Most teams survive this because these cases of many phantoms in one frame are reasonably rare. However, they do pose a very serious threat to the correctness of their overall play as this can result in a fatal fault of the software and thus a shutdown of the robot while in 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.

4.2 Modeling with Plausible Logic

We introduce the modelling of the above example incrementally. We start with a simple example but the point is not only to help the understanding of using non-monotonic logic, but to describe the path chosen for implementation on the Sony AIBO robots used by the MiPal 2005 Griffith team at RoboCup. We illustrate that higher level models can be introduced incrementally as extensions/refinements of the previous model.

At the time of the competition, 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). 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. This had to be extended by automatic production of C++ macros, automatic production of gluing code and a template method (so that the logic would execute on the Sony AIBO).
Model 1

This is a simple model presented here to illustrate the entire process, it includes introducing the logic programming language as well as the gluing C++ code. The model provides correct results only if the vision module reports exactly one landmark or none. However, as new more sophisticated models are introduced, only the model changes and the entire process (even the template method to execute the consistency module on the robot) remains unchanged. Later models require more off-board CPU-time, as the process builds proofs for the expressions of interest and then heuristically finds simplifications of those expressions. Those expressions would be more elaborate as well. However, all the models discussed here were executable on the Sony AIBO ERS-7 with very reasonable CPU-time requirements thanks to this architecture.

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 ∪ PostType. In the programming language we have

type Landmark = GoalType + PostType.

The next step is to define the inputs (what vision reports) as predicates. In this case, each input would be 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 ⇒ p and ¬a ⇒ ¬p, where p is the plausible assumption to be fired by a. A macro simplifies this.

$\texttt{declareInput$(a, p)\{\$# input{$+a$.} {+$a$.} => {$+p$.}. {+$\neg a$.} => {$\neg p$.}. $\}$#}$

Now, we can write the plausible assumptions. First, See(x) means vision reports seeing the landmark x.

type See(x <- Landmark).

We will construct C++ glue code so that 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.

$\texttt{declareInput$($\text{FG}$,See($\text{FG}$))}

Similarly, we have five more declarations.

$\texttt{declareInput$($\text{BG}$,See($\text{BG}$))}$
$\texttt{declareInput$($\text{LP}$,See($\text{LP}$))}$
$\texttt{declareInput$($\text{RP}$,See($\text{RP}$))}$
$\texttt{declareInput$($\text{RBP}$,See($\text{RBP}$))}$
$\texttt{declareInput$($\text{LBP}$,See($\text{LBP}$))}$

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. In PL this is R1 : {} ⇒ ¬CS(x) while in the programming language DPL we write

R1: ⇒ ¬CS(x).

However, if vision reports a landmark, we believe it. PL writes this as $R2 : \text{See}(x) ⇒ CS(x)$ $R2 > R1$. Note the relationship between rules. Now, the DPL equivalent is

R2: See(x) => CS(x). R2 > R1.

This works because we also provide a rule that any frame that has two or more landmarks should be ignored.

The programming language DPL allows us to request which outputs we want to build proofs for. In what follows p means the π (plausible) level of proof.

output(p CS(FG),"CS_FG"). output(p CS(BG),"CS_BG"). output(p CS(LP),"CS_LP"). output(p CS(RP),"CS_RP").

output(p CS(RBP),"CS_RBP"). output(p CS(LBP),"CS_LBP").

This completes the programming of this simple model. The model is executed in DPL (outside the robot) and as a result we obtain a header file in C++ named roboConsistency.h.

To complete the description of how PL is used in theSony AIBO, we present here the C++ code of the template method in the consistency module. The template method is executed every time a new frame arrives and the vision module is reporting about landmark sightings in the frame.

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

In particular, the three macros used in here 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 codes for a visual testing tool. The code in ConsistencyMacros.h is computer generated, and although simple, we omit it here for space reasons. Suffice to say that INIT_ALL_FALSE() is a macro that creates the necessary definitions of C++ Boolean variables for all landmarks. UPDATE_ALL() queries the reports of the vision module. For example, a variable for the front goal previously initialised to false may now be set to true if vision has found a front goal in the current frame. It also provides a pointer to such an object so other attributes about the landmark can be evaluated, like its size or if it is to the left or right of another landmark in that frame. Finally, PLACE_CS_ALL(); will evaluate the expressions for which we requested outputs and if they evaluate to true, it will forward such landmark sighting to the localisation module (or any other module that may benefit from it, like the behaviour to kick when the front goal is visible). In particular PLACE_CS_ALL(); will have as many if statements as outputs requested. The expression of the if statement is defined in the macros in roboConsistency.h. For the particular case of the Model 1 just presented, testing (evaluating) the CS_FG macro is simply the boolean value of the variable FG. 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 localisation. 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, resulting in localisation receiving the sighting information about the front goal.

Higher Level Models

While Model 1 is simple, it allows validation of the entire concept of a non-monotonic logic implemented on a Sony AIBO with the actual reasoning performed off-line and transporting pre-compiled logical proofs to the robot. This avoids the execution of the inference engine on-line while the Sony AIBO is processing images and carrying out all the other processes for its performance as a soccer playing robot.

We now introduce progressively more sophisticated models. Following the lines of a previous discussion 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 a bit more the domain. In particular, we use Opp(x, y) to mean x is opposite y. Also Opp(x, y) iff Opp(y, x). This appears in the programming language as

type Opp(x <- Landmark, y <- Landmark - {x}).
default ~Opp(x, y).


Also, we want 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).


We are now in a position to use inputs from vision (using the same macro to declare a rule and its contra-positive). Declarations for See(x) and axioms relating this predicate to the C++ expression FG are as before. We use the command

ignore 3 of {"FG", "BG", "LP", "RP", "RBP", "LBP"}.
so the implementation of PL will eliminate building proofs for settings with 3 or more landmarks. This is just a device to save CPU-time on the off-line computation of expressions for the robot.

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. With a macro for an axiom LP_FG that fires SeeLtoR(LP, FG) allows not to 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.

R3: {See(x), See(y), Opp(x, y)} $\Rightarrow$ ¬Cs(x) R3 > R2
But, if vision reports two objects out of left-to-right order, then we believe neither.

R4: {See(x), See(y), SeeLtoR(y, x), LR(x, y)} $\Rightarrow$ ¬Cs(x)
R4′: {See(x), See(y), SeeLtoR(y, x), LR(x, y)} $\Rightarrow$ ¬Cs(y)
R4 > R2, R4′ > R2.

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

R1: => "Cs(x). R2: See(x) $\Rightarrow$ Cs(x). R2 > R1.
R3: {See(x), See(y), Opp(x, y)} $\Rightarrow$ "Cs(x). R3 > R2.
R4: {See(x), See(y), SeeLtoR(y, x), LR(x, y)} $\Rightarrow$ "Cs(x).
R4: {See(x), See(y), SeeLtoR(y, x), LR(x, y)} $\Rightarrow$ "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 localisation module (by R1) but then conclude that there is a landmark sighting to be forwarded (because of R2). However, it may change that conclusion in light of R3. If we want to add rules like in the model of the previous section that used other aspects in the report from vision (to report a post over a goal, or the largest of two goals), now this is easily achievable.

We will see in the next section that there is a case with 3 landmarks in a frame where this model could be refined. To describe this new refined model we have to state some more facts about the environment. We need to say that Adj(x, y, z) means x is left of and next to y, and, y is left of and next to z. In DPL this is as follows.

type Adj(x <- Landmark, y <- Landmark - {x}, z <- Landmark - {x, y}). default ~Adj(x, y, z).

We consider 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 R4) the mutual consistency of y and z are grounds
for overriding the conclusion by $R_4$. This leads to an extension of Model 2 that we name **Model 3**.

\[
\begin{align*}
R_5 : \{\text{See}(x), \text{See}(y), \text{See}(z), \text{SeeLtoR}(y, z), \\
\text{SeeLtoR}(z, x), \text{Adj}(x, y, z)\} & \Rightarrow Cs(y). \\
R_5 : \{\text{See}(x), \text{See}(y), \text{See}(z), \text{SeeLtoR}(y, z), \\
\text{SeeLtoR}(z, x), \text{Adj}(x, y, z)\} & \Rightarrow Cs(z).
\end{align*}
\]

Similarly, if $z$ is the one out-of-order, believe $x$ and $y$.

\[
\begin{align*}
R_5 : \{\text{See}(x), \text{See}(y), \text{See}(z), \text{SeeLtoR}(z, x), \\
\text{SeeLtoR}(x, y), \text{Adj}(x, y, z)\} & \Rightarrow Cs(x). \\
R_5 : \{\text{See}(x), \text{See}(y), \text{See}(z), \text{SeeLtoR}(x, z), \\
\text{SeeLtoR}(x, y), \text{Adj}(x, y, z)\} & \Rightarrow Cs(y).
\end{align*}
\]

The rules that complete this model are as follows.

\[
\begin{align*}
R_6 : \{\text{See}(x), \text{See}(y), \text{See}(z), \text{SeeLtoR}(x, z), \\
\text{SeeLtoR}(z, y), \text{LR}(x, y), \text{LR}(y, z), \text{Opp}(x, z)\} & \Rightarrow Cs(x). \\
R_6 : \{\text{See}(x), \text{See}(y), \text{See}(z), \text{SeeLtoR}(x, z), \\
\text{SeeLtoR}(z, y), \text{LR}(x, y), \text{LR}(y, z), \text{Opp}(x, z)\} & \Rightarrow Cs(y).
\end{align*}
\]

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

### 5 Experimental evaluation

The implementation of these PL models has been evaluated in two directions. First, the algorithms to build the proofs off-line with the implementation of PL in Haskell have been analysed for the purposes of their computational requirements. Second, we have analysed the effectiveness of the approach in an ERS-7 Sony AIBO in the lab and in the actual RoboCup 2005 competition.

It is of interest that although a major computational effort is performed off-line, the expressions obtained for the proofs of the output predicates are significantly larger as the modules become more complex. For example, for Model 2 the file `roboconsistency.h` has 4700 tokens for the 6 expressions that determine the consistency of the landmarks. For Model 3, this file has 12,530 tokens. However, these expressions are in conjunctive normal form and the terms are literals of boolean variables. The C++ compiler can optimise their evaluation, and in fact we profiled the execution times on the Sony AIBO ERS-7. Evaluating all 6 expressions in Model 1 required only $44 \pm 1$ microseconds (this is at a 99% confidence interval). Note that Model 1 has trivial expressions in `roboconsistency.h`. Evaluating all 6 expressions in Model 2 required only $60 \pm 1$ microseconds (this is also 99% confidence interval) and Evaluating all 6 expressions in Model 3 required only $110 \pm 3$ microseconds (also 99% confidence interval). However, in all three models the constant overhead of initialising the boolean values and retrieving the pointers from the vision module was $30 \pm 1$ microseconds. This means that the evaluation of the large expressions is comparable to the initialisation of a few boolean variables and in fact orders of magnitude faster than the processing of a frame by the vision module or the processing of a localisation update (the revision of the position belief because of sensor inputs and action) in the localisation module.

We now discuss the effectiveness of the approach. The entire architecture was also validated by a Graphical User Interface (GUI) to validate a setting with respect to the model and by evaluating the implementation also on the Sony AIBO. That is, the `roboconsistency.h` files produced for a model as output by DPL can be used by the GUI or by the Sony AIBO.

A user interacting with the GUI 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 to the right of the front goal). The GUI tool then indicates which of these landmark sightings are regarded as worth forwarding for localisation. This GUI tool facilitates the debugging of the entire architecture without having to execute the consistency module on the robot. Of course the second part of this validation consists of executing the model in the consistency module of the robot. Again, this is just a compilation option that builds the binaries for the memory stick of the Sony AIBO using the corresponding `roboconsistency.h` files.
We evaluate the results on the robot with a telnet connection that displays the ID of the objects reported by vision (see Figure 1 and Figure 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 localisation. For Figure 1, Model 2 provides the correct result. Namely, we can forward to localisation the left post, but neither the goal nor the right post should be forwarded to the localisation 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 modelling with PL as opposed to modelling without it. The PL is analysing 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. Not all settings with 3 landmarks are correctly identified with Model 2. Figure 2 is the 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.

With the GUI and the telnet evaluation, the models were evaluated for all possible configurations of sightings (phantom combined with real) that involve three or less landmarks. Model 3 was used in the Mi-Pal 4-legged league team at RoboCup 2005. The following figures show images at the venue in Osaka where the consistency module filtered phantom objects for localisation. Figure 3 shows the functionality of the vision system on board the Sony AIBO. We have enlarged the captured image on board, then the blobs of the colour 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. Figure 3 shows that the blue match score and timer appears 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 4 shows that phantom sightings occur even with the regular colour-coded objects in the field. The ball has enough yellow pixels to be confused for a yellow goal against a blue goal. Figure 5 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.

6 Final remarks

The models presented above are strong enough to be useful at RoboCup 2005. In order to construct more complex models we are porting the reasoning engine to C++ so it can run on the Sony AIBO. We expect that in this way, the robot would be able to reason about more objects and we would trade the fact that proofs are pre-built for all possible questions with only performing online those proofs that are actually needed. We are also currently working on more complex modelling that will include time, thus reasoning could be performed amongst sightings in different frames taken at different times; perhaps consecutive frames.

We have described the use of non-monotonic reasoning for making judgements on the landmarks identified by a
vision module. This has been achieved by the use of PL and an architecture that allows the construction of all necessary proofs off-line (pre-computed). The proofs are also simplified off-line thus the robot only needs to evaluate a logic expression that establishes the logic value of the required predicate. The usability of this approach has been demonstrated as a filter for localisation in the RoboCup 4-legged league. As the RoboCup organisation and its leagues continue to evolve the rules of the competition, they have progressively introduced natural lighting. This aspect alone results in more false positives and true negatives reported by vision when identifying landmarks. Thus, reasoning about sightings with respect to domain knowledge will become more necessary in order to judge which are reliable enough to incorporate into a localisation module and which are best left out.

References


