RDRVision - Learning vision recognition with Ripple Down Rules

Kim Cuong Pham , Claude Sammut
School of Computer Science and Engineering
University of New South Wales, Australia
{kimpham,claude}@cse.unsw.edu.au

Abstract

Most approaches to robot vision require domain knowledge to be programmed. Such programs are expensive to create and usually fail to adapt to new or changing environment. In this paper, we investigate methods for learning domain knowledge in object recognition tasks. We use Ripple Down Rules, a rule-based representation that can be learned incrementally. The method is demonstrated in the vision tasks required in the RoboCup 4-legged robot soccer competition. Current limitations as well as future improvements are discussed.

1 Introduction

In the RoboCup robot soccer competitions, objects on the field are usually distinguished by their colour. Often, however, the relationship between colour blobs is significant. For example, the top left hand image in Figure 1 shows one of the four beacons placed around the field to help robots localise. To the robot's vision system, the beacon will consist of two blobs, blue and pink. The goals are also coloured, one of them, the same blue as contained in the beacon. To distinguish the different kinds of blue bobs, we generally write rules such as "if the blue blob is above the pink blob, it is part of a beacon; if the blue blob is above the green field, it is the goal". Unfortunately, the life of a soccer playing robot is not that simple because these rules must be qualified depending on orientation of the head, whether part of the image is occluded and the fact that different colours can appear similar to each other under different lighting conditions. As a result, object recognition rules can become extremely complicated and difficult to modify.

The motivation for the work presented here is to find a way of creating domain specific object recognition rules in a simpler and more systematic manner. We do this by applying a form of supervised incremental learning based on Compton's Ripple-Down Rules (RDR) [Compton and Jansen, 1988]. We contend that the construction of domain specific object recognition rules is common to many robotics applications. We demonstrate our method in the context of the RoboCup 4-legged robot competition but we believe the method is more widely applicable.

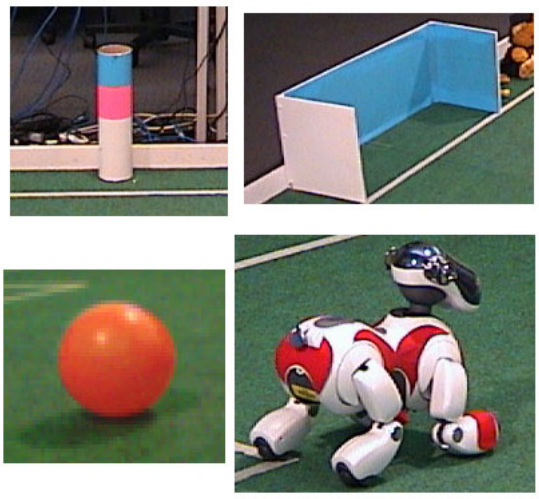

Figure 1: Object of interest in RoboCup Soccer

In the remainder of this paper, we describe the RoboCup vision setup. We then introduce Ripple-Down Rules and how they can be used to learn to recognise objects. The learning system has been implemented in a Java tool that builds object recognition rules that can be exported to an Aibo. We report on preliminary experiments that indicate that this method can, indeed, simplify the task of acquiring domain specific visual knowledge.

2 The Vision System

The vision systems in the Aibo soccer robots are becoming increasingly sophisticated as improvements processing power allow more complex operations to be performed. Several of the leading teams use combinations of

colour and edge detection also using some knowledge of the expected shapes of objects. For simplicity, we have chosen to use colour as the primary low-level feature in these experiments. The vision system [Chen et al., 2003] consists of the following components:

1. Colour Segmentation: converts a YUV colour image into a segmented image, containing seven predefined colours (e.g. orange for the ball, pink for one beacon component, etc).
2. Blobber: a connected components labelling algorithm that creates blobs of the same colour. [Fisher et al., 2003].
3. Object Recognition: filter and process blobs to identify objects. This is divided into two parts:
   - Object Finding: Discard spurious blobs using hand-coded rules.
   - Sanity Checks: discard objects that are inconsistent with some prior knowledge.

Since it first entered the RoboCup competition in 1999, the rUNSWift team has used machine learning to build the lookup table used in colour segmentation. Calibration of the lookup tables is done by taking sample images manually labelling pixels using a painting program. The labelled data can then be used to train a classifier. Several types of classifiers have been tried, including decision trees, nearest neighbour and, in 2005, ripple-down rules. Once the pixels are labelled, a blob forming algorithm is run. The only thing that distinguishes this phase of the process is that the algorithm must be as efficient as possible as this is one of the most time-consuming operations for the vision system.

The object recognition components of the vision system are entirely hand-coded. Over the years of competition, this code has become complex and arcane as new elements are added to deal with previously uncovered cases. As in any large-scale knowledge engineering exercise, maintaining consistency of the code is one of the most difficult aspects of implementing modifications. As we shall see in the next section, ripple-down rules were designed precisely for the purpose of making knowledge-base maintenance relatively easy and painless. We first describe generic ripple-down rules and how they are acquired and then we examine how they can be adapted for use in a vision system.

3 Ripple Down Rules

Ripple-down rules are nested "if" statements that also contain exception conditions. They follow the general form:

```
if <condition> then <conclusion>
    except if ...
else if ...
```

An RDR is executed in the same way that a normal "if" statement would be with the addition that when a condition evaluates to true, before returning the corresponding conclusion, we first check the exception and only return the conclusion if the exception fails. That is, exceptions override their parent rule.

Ripple-down rules are created as follows. Initially, the RDR consists only of a default rule:

```
if true then <default conclusion>
```

Each time a new case is encountered that is incorrectly classified, the RDR is updated to correctly cover that case. Thus, if:

- the new case is misclassified, that is, it satisfies a condition that should not, an exception is added to the rule whose condition was satisfied. The exception condition must distinguish the new case from the old.
- the new case is not able to be classified, an alternative or "else" branch is added to the last rule whose condition was satisfied.

Another way of understanding the update process is: when the RDR is too general, it is specialised by adding an exception; when it is too specific, it is generalised by adding an alternative.

To apply RDRs to visual processing, we must define how generalisation and specialisation work with numerical attributes. For example, each pixel is represented by the YUV values delivered by the Aibo camera. Y is the intensity of the pixel and U and V are the colour components. All values are in the range 0 to 255. To operate on numerical ranges, we use Scheffer's variant of RDRs, called Cut95 [Scheffer, 1996].

An example is represented by a tuple of n attributes $< x_1, x_2, .., x_n >$. The condition part of a rule is a set of intervals $[m_1, M_1], [m_2, M_2], .., [m_n, M_n]$, where m_i is the lower bound and M_i is the upper bound of attribute x_i. For example, a pixel, represented by a set of YUV values will only satisfy a rule's condition if all three values fall within the specified intervals. The set of intervals can be visualised as a hyper-rectangle in high dimensional space and the example as a point in that space. The example satisfies the condition if the point is contained inside the hyper-rectangle. The entire RDR becomes a set of hyper-rectangles, some of them are nested. The Cut95 algorithm for incrementally updating a RDR is shown in Figure 2.

This algorithm was implemented by the second author and was used by the rUNSWift 2005 RoboCup team to build the lookup tables for colour segmentation. The program provides an interactive graphical environment for labelling pixels. Following our positive experience

```
Step 1: Find the smallest hyper-rectangle
    that contains the new example.

Step 2: Update the rule as follows:

    if the classes of the example and this
        hyper-rectangle are the same then
            do nothing
    else
        if the exception is empty then
            create new exception
        else if the hyper-rectangle can be
            extended to cover this point then
                extend it
        else
            create a new alternative.
```

Figure 2: RDR Update rules

with this method for learning low-level concepts, the question then arose as to how suitable ripple-down rules would be to learning higher-level classification rules for object recognition. The first stage in investigating this proposition was to devise a set of attributes suitable for representing useful features and relations between objects.

4 Vision Features Selection

As our low-level vision system is based only on coloured blob detection, all higher-level features must be derived from blob attributes. An important requirement of the features is that they should be invariant, to camera translation, scale, and rotation. Some objects consist only of single blobs, such as balls and goals. Other objects are composed of several blobs, for example, each beacon contains a pink blob and either a blue of yellow blob. Thus, we create a set of features to characterise a single blob and another set to represent relationships between bobs.

The features that we use to characterise single blob objects are:

- Aspect Ratio: the ratio between the bounding box[1]width and height.
- Circularity: the ratio of the area divided by the square of the perimeter.
- Density: the ratio between the area and the bounding box. This is helpful in identifying "noise" blobs since they are usually "sparser" than real ones.
- Relative height: Geometrically, the real height of the object from the ground, is proportional to $sin(elevation)$ and disproportionate to the height of the object. Therefore, the ratio of these two figures would give the relative height of the object (Note that the absolute height depends on the actual size of the object, which we do not know).

In order to recognise multiple-blob objects such as beacons, we have to categorise relationships between blobs. For these experiments, we only deal with beacons, which consist of exactly two blobs. The other multiple-bob objects are the other robots on the field. We do not deal with these here as they were too complicated to deal with in the time available for this work. Note also that we do not attempt to handle multiple blobs when objects are partially occluded.

The features for two blob relations are :

- Gradient: the angle between the line connecting two blobs' centroids, and horizontal line. This characterises how tilted the object is. It can eliminate spurious objects whose orientation is not vertical.
- Blob Ratio: the ratio between the sizes of the two blobs. Size can be area, width, or height. We chose the maximum of width and height as it is more robust to occluded beacons.
- Centroid line covering: we count the number of pixels on line between the centroids of the two blobs. This characterises how far apart the blobs are. If the two blobs are close together, all pixels will be covered by the two blobs, otherwise, there will be pixels in between the two centroids that belong to other blobs or the background.

Clearly, we have engineered the features to suit the requirement of this particular soccer field. Future work will be aimed at providing tools for automating feature selection.

Object recognition occurs in two steps. We first attempt to recognise single blob objects before multiple-blob objects. The advantages are two-fold. Firstly, searching for beacons requires $O(N^2)$ relations to be examined, where N is the number of single blob candidates. By filtering single blobs, N is reduced significantly resulting in faster execution. Secondly, the two step process gives more flexibility to users when building rules, as we shall see in the following section.

5 Implementation

Domain specific object recognition rules are acquired incrementally by supervised learning. A graphical tool is used to present training instances to a human. The system attempts to classify the objects in an image and

[1]Since we know the pose of the camera (from robot kinematics), we can transform objects so that their orientation is relative to the ground, hence their bounding box is invariant to camera orientation.

then shows these to the trainer, who has the option to correct any misclassifications. Each time a correction is made, the RDRs are updated so that subsequent similar objects should be correctly classified.

The RDR Colour program, or simply RC, began the training system for colour classification. It has since been extended by the first author to build RDR object recognition rules (Section 5.1) and to actively acquire new knowledge by interacting with robots. We first describe the off-line training for object recognition and then the online robot interaction.

5.1 The Training Tool

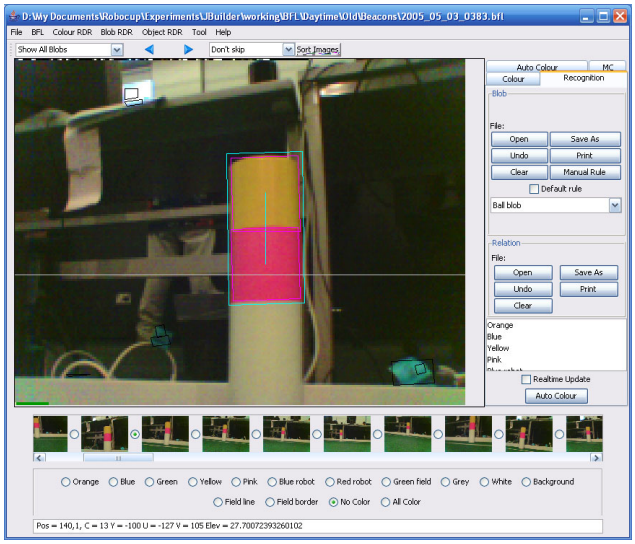

Figure 3: RC Tool

Figure 3 shows a typical screenshot of RC. It can display images either from a batch of examples, selected from the movie strip, or live images streamed from the robot. In this illustration, an image has been selected from the movie strip.

The program draws a pink bounding box around objects that are recognised by the current RDR. Black bounding boxes surround blobs that are not recognised. When trainer user clicks within the bounding boxes, he or she can change the classification to cause an RDR update.

The cyan bounding box indicates a multiple-blob classification. The centre line shows the distance between the centroids of the component blobs. If the program fails to recognise two related blobs as belonging to the same object, the trainer can drag one blob onto the other to indicate that they are related. if the program incorrectly shows a relationship when there is none, the user can click on the cyan box and indicate that that relationship in is invalid, causing another RDR update. The RDRs can be saved to an XML file.

A part of a goal RDR is given in Figure 4. Initially, the RDR contains only one default rule. After some positive examples are classified, it created an exception rule covering all those examples. The RDR is then over-generalised, thus false positive examples are encountered. Figure 4,c shows the RDR after correction, with two new exceptions created. Note that the actual goal RDR are much larger than this example.

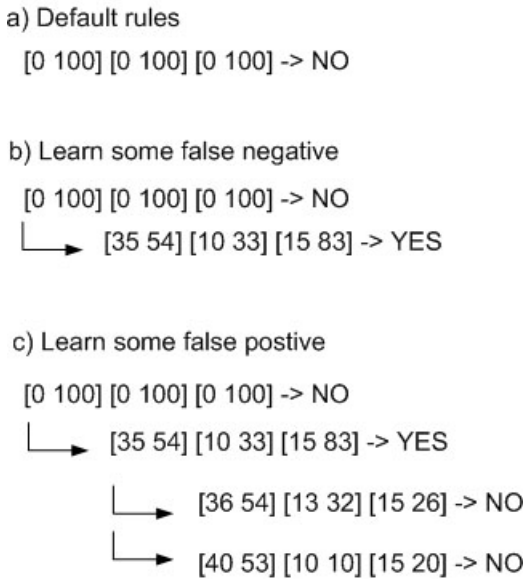

Figure 4: Sample RDR Updates

As well as being able to train RDR offline and transfer the rules onto robots, RC can also update rules in real time. Every time a new example is added, it is sent to the robot over wireless. The robot, which also runs a RDR engine, updates its rules according to the new example. In this way, the tool can interact with the robot in various ways described in the next section.

5.2 Robot Interaction

One advantage of incremental RDR learning is that it allows a robot to actively search for new training instances. These are images that contain objects that are inconsistent with knowledge base. By searching for these examples, the robot is verifying the current rules and extending them on the fly.

Training instances can either be *false positives* (objects that are incorrectly classified) or *false negatives* (object that are not recognised at all). *False positives* are easy to look for if we set up training so that we teach the robot how to recognise one object at a time. We put a robot on the field, hide the target object and ask the robot to send any images that contain the target object. Those images must contain a false positive.

Similarly, *false negative* examples can be collected by asking the robot to send images that do not contain the target object. However, if the robot sends every negative image, we may be flooded. To reduce the potentially large number of images, we adopt a heuristic that only the first false negative in a sequence of false negatives is sent. This greatly simplifies training while still having a high probability of providing sufficient information for efficient learning. Video clips of the training process are available at [Movies, 2005].

6 Evaluation

The RDR vision system was evaluated on the RoboCup in the rUNSWift laboratory. The surrounding environment was unstructured and spurious objects were placed on the field. Figure 5 depicts some parts of the environment.

Figure 5: Top left: spurious objects on tables; Top right: orange walls; Bottom: robots and material in the surrounding area

The goal was to have robots reliably recognise objects on the field, regardless of what is in the surrounding environment. Note that some of the walls of the lab are the same colour orange as the ball. The performance of the system was measured by the training time required and by its recognition accuracy.

6.1 Training Time

The time complexity of machine learning algorithms is usually measured in terms of the CPU time taken to execute a program. However, in robotics, the more important figure is the how long it takes to collect training examples. In our system, the robot can actively search for training examples. This greatly simplifies the process and makes knowledge acquisition relatively quick.

Object	Ball	Goal	Beacon Blob	Beacon
Time (minutes)	20'	11'	15'	
Examples	77	43	49	14
Positive ex'	45	25	49	14
Negative ex'	32	18	0	0

Table 1: Training result

Table 1, shows the amount to time taken to collect examples and to train the system to recognise the ball, both goals and the four beacons. Beacon training rules consist of single blob rules and blob relation rules. Their training times are the same as they are trained together. The total time for getting the whole system up and running was approximately 45 minutes. Although it is difficult to make an objective comparison, these times are best compared with the amount of time it would take to manually program and test similar code. We believe the latter would be significantly longer.

Beacon single blobs and composite beacon objects were trained at the same time. However, the number of examples needed for single blobs was much larger than for the relations between blobs. This suggests that the metrics used to characterise relations are more discriminatory than those for single blobs. Also note that there were no negative examples for beacons. This demonstrates that the RDR generalisation of the beacon description was very conservative and never overgeneralised to the point that it incorrectly classified a beacon.

6.2 Accuracy

In order to evaluate the recognition accuracy of our system, we put the robot in five different locations on the field. The robot was made to spin on the spot and to rotate its head. It was then instructed to send all images that contain a target object (for false positive detection) and all images that do not contain the target object (for false negative detection). The images received from the robot, were check to see if the objects were correctly classified and a count of misclassifications was kept. Figure 6 summarises the results.

Note that due to the low resolution of the camera distant objects may occupy only a few pixels in an image and so classification can be unreliable. When evaluating balls and beacons, we only counted positive images that contained objects within two meters of the robot. However, this is far that many spurious objects off the field are still within range.

The average error rate for the ball was 11%, for the goal 12.4% and for the beacons, 6%. The *false negative* error is larger than *false positive* error, most clearly for beacon recognition (8.5% to 3.2% on average). A large proportion of the misclassified images have objects on

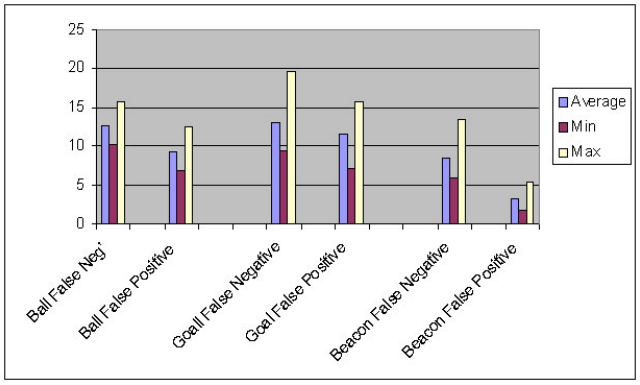

Figure 6: Recognition Error

the edges of the image or are occluded. This indicates that our metrics were not tailored well enough for these objects conditions.

6.3 Ball recognition

Our second experiment was to discriminate the ball from various spurious objects. We put a number of objects with a similar orange colour on the field. Figure 7 shows the objects that we used to try to fool the robot. The surrounding area was the same as in the previous experiment and also contained more spurious objects. In this experiment, made use of the *circularity* metric. Furthermore, if an image contained a ball, we only use it if the ball is within 1.5 metres from robot. This ensures that lighting conditions have little influence on the evaluation.

Figure 7: Counterfeit objects and their segmented image

Table 2 shows the result of the experiment. 89.15% percent of ball instances were recognised correctly. Only a few pen and sign instances were incorrectly recognised as balls (4.11% and 2.94% respectively). While no teddy bears were recognised as ball, the duck turned out to be very difficult to differentiate from the ball. It has exactly the same colour as the ball and similar rounded

	Ball	Pen	Bear	Duck	Sign
Examples learned	31	8	14	35	7
Testing images	212	73	81	107	68
Recognized as ball	189	3	0	35	2
Percentage	89.15	4.11	0.00	32.71	2.94

Table 2: Recognise ball against ball-similar objects

contour. In fact, in some viewing angles, the duck, after segmentation, looks exactly like the ball.

6.4 Localisation Challenge

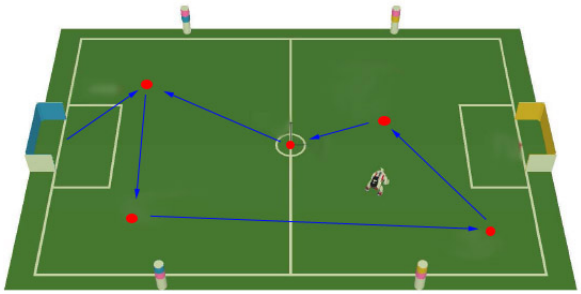

Figure 8: Localisation Trajectory

The final evaluation is an integrated test. Based on the localisation challenge set in several RoboCup competitions, we instruct the robot walk through the five points marked in Figure 8. The robot stops at each point and wags its tail before moving on to the next point. The error is measured by how far the robot is from the designated points. The robot performed the task ten times and the results were averaged and shown in Figure 9.

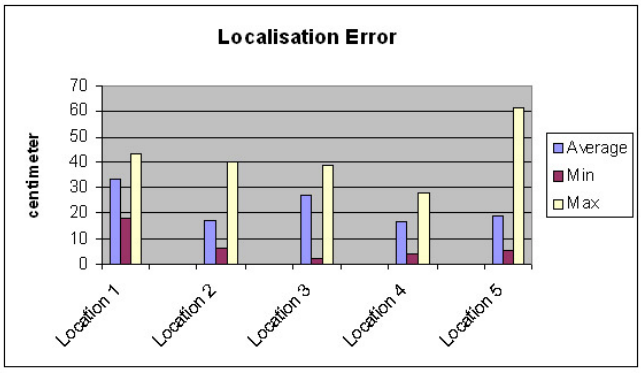

Figure 9: Localisation Error

On average, the robot can localise within a 25 cm error. This is comparable to the robot's performance in the RoboCup competition, using hand-crafted vision code.

On average, there is no noticeable performance difference between locations on the field. There was only one run where the robot stopped early from location 4 to location 5, resulting in an error of 80 cm from the point.

7 Conclusion

The system described in this paper is capable of acquiring domain specific object recognition rules that can run in real-time on a robot with modest hardware. It's greatest attraction is the ability to learn incrementally, that is, it can update its knowledge base as each new training example is obtained. This is particularly useful in facilitating active learning. This kind of knowledge acquisition system promises to reduce the time and effort required to build vision systems that depend on domain knowledge.

The present system is only a proof of concept. It has many limitations that must still be overcome. Its performance is still heavily dependent on well chosen features. As we saw in the experimental results, the features chosen here, failed to recognise objects when they were occluded or on the edge of the image. They were insufficient to distinguish objects such as the ball and the toy duck. They were not sufficiently reliable beyond a distance of 1.5 to 2 metres.

A more promising approach to be pursued in future work is to expand the expressiveness of the RDR rules so that a library of background knowledge can be called upon to select appropriate features. Also, the present ability to express relations is extremely limited and inflexible. A more expressive language for representing conditions need not be restricted to a fixed number of tests but can combine them in an arbitrary number of conjuncts.

Acknowledgments

The authors thank Will Uther and the RoboCup team members their assistance and for finding bugs in the code. We also thank Bernhard Hengst for some motivating discussions later in the work.

References

[Chen et al., 2003] Chen, J., Chung, E., Edwards, R., Wong, N., Hengst, B., Sammut, C., and Uther, W. (2003). The unsw robocup 2003 legged league team. Undergraduate thesis in computer and software engineering, The University of New South Wales.

[Compton and Jansen, 1988] Compton, P. J. and Jansen, R. (1988). Knowledge in context: A strategy for expert system maintenance. In *Proceedings of the Australian Artificial intelligence Conference.*

[Fisher et al., 2003] Fisher, R., Perkins, S., Walker, A., and Wolfart, E. (2003). Hypermedia image processing reference: Connected components labeling. http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm.

[Movies, 2005] Movies (2005). Illustrating video clips. http://www.cse.unsw.edu.au/%7Ekimpham/papers/.

[Scheffer, 1996] Scheffer, T. (1996). Algebraic foundation and improved methods of induction of ripple down rules. In *In proceeding of Pacific Knowledge Acquisition Workshop.*