A Descriptive Language for Flexible and Robust Object Recognition

Nathan Lovell and Vladimir Estivill-Castro

School of CIT, Griffith University, Nathan 4111 QLD, Australia

Abstract. Object recognition systems contain a large amount of highly specific knowledge tailored to the objects in the domain of interest. Not only does the system require information for each object in the recognition process, it may require entirely different vision processing techniques. Generic programming for vision processing tasks is hard since systems on-board a mobile robots have strong performance requirements. Such issues as keeping up with incoming frames from a camera limit the layers of abstraction that can be applied. This results in software that is customized to the domain at hand, that is difficult to port to other applications and that is not particularly robust to changes in the visual environment.

In this paper we describe a high level object definition language that removes the domain specific knowledge from the implementation of the object recognition system. The language has features of object-orientation and logic, being more declarative and less imperative. We present an implementation of the language efficient enough to be used on a Sony AIBO in the Robocup Four-Legged league competition and several illustrations of its use to rapidly adjust to new environments through quickly crafted object definitions.

1 Introduction

Most object recognition systems use hard-coded, domain specific knowledge. For example, all leagues in Robocup rely on the ball being orange and spherical. If it were changed to be a non-uniform colour or a non-uniform shape then most object recognition systems would have to be largely re-coded[1]. We saw how devastating this was in last years challenge in the four-legged league which required the robots to locate a black and white ball where only eight of the twenty-four teams managed to even identify the ball and no team passed the challenge[2].

This paper presents a higher level descriptive language of the objects we are likely to recognise in a particular domain and thus leave the underlying vision

[1] This is certainly true for the team Griffith 2003 code. There are many other examples of systems that are programmed in this way because they are based on the vision algorithms developed by Carnegie Mellon University[1] for the year 2000 competition.

[2] http://www.openr.org/robocup/challenge2003/Challenge2003_result.html

D. Nardi et al. (Eds.): RoboCup 2004, LNAI 3276, pp. 540–547, 2005.

system programming unchanged. We claim that the approach allows rapid expansion to newer or changing vision domains. The case study for this work is the Sony AIBO platform in the Robocup Four Legged league. By modern standards in vision processing systems, these inexpensive robots are not computationally rich. We will present both the high-level descriptive language called XOD (XML Object Description) and the techniques used to translate it to C++ code. We will also present several illustrations where we have used our system to recongise vastly changed objects on the soccer field.

2 Related Work

There have been a considerable number of studies on different methods for representing objects generically in such a way that they can be located within target images. Much of this work focuses on trying to learn object representations from sample images and, as such, the generic representation usually takes the form of a statistical model [2] or some subspace representation of the features of the object [3]. These methods rely on supervised learning techniques. While these systems do function reasonably well in variant lighting conditions and poses of objects, they are still not easily applied to the task of mobile robotics. They work well in situations where the object, though possibly unknown, is in a controlled environment within the image (such as when the background of the image is known and the object features, therefore, can be easily extracted) but are less tolerant to object occlusion and background uncertainty.

Another closely related problem, which our work also addresses, is that of how to combine and apply existing vision processing techniques to the object recognition task. For example, some object A may require that edges be extracted and the texture of the internal pixels analysed for identification while another object B is more easily identified by colour segmentation and connected region analysis. Draper [4] proposes that the task of object recognition is a goal driven task and the user of the system should not need to specify which combination of vision processing techniques are applicable on a per-object basis. Instead Draper proposes that the vision processing tasks themselves can be treated as primitives and their correct combination and application learned by the system for each object, again by supervised learning techniques. Our work addresses this problem by the static transformation of user-created object descriptions into a vision processing pipeline for each object which will use only applicable techniques according to the description.

The difference between the problems discussed above and the problem addressed by this paper is that in the field of mobile robotics one usually has the added difficulty of determining *if* any of some known set of objects is present in an image and, if so, *where*. Our work simplifies the task of describing the objects - our system does not learn object descriptions but rather they are given to us by the user. The end of this is similar to other systems in that the domain specific knowledge is still removed from the vision processing module and the details of the vision processing itself are not needed to be understood by the user. The ad-

vantage of this approach, however, is that our system allows for quick and robust adaptation to varying visual domains with no prior sample images. Supervised machine learning techniques require large databases of classified samples and substantial amounts of processing time, neither of which is necessarily available in the field of mobile robotics.

The idea of codifying the domain specific knowledge in a goal driven and code independent manner is not new. The German team in Robocup 2003 [5] presented an XML based specification language for agent behaviors. Our paper presents a similar system applied to the task of visual object recognition.

3 The XOD

XOD stands for "XML Object Description" and it is the first version of a language used to describe the objects we expect our robot to see in a code-independent way. The language works with several types of primitives - points, lines, blobs and objects - as well as collections of these primitives. When we formalise the language as a logic where these primitives are the terms of the logic. The relationships between the primitives of our language become the predicates of the logic. We also take advantage of the overlaps between object-orientation and knowledge representation in AI to allow our primitives to have properties defined on them. We now define what we mean by these primitives.

A *point* is used to represent a pixel. It is not the responsibility of the vision system to convert items from pixel into world coordinates so a point's location (x, y) in an image is one of its properties. We can use points to specify properties of other objects such as line intersections or centers of objects.

A *line* is simply a collection of connected points, for example a connected border between two different colours. There are two stages of processing on lines - edge detection and vectorisation. It is sometimes useful to represent lines in raster form and sometimes useful to represent them in vector form so our language allows descriptions for both representations. Our language operates on the properties of lines and the relationships between lines and blobs to find objects. Note that a property of a line could also be an object (as in the line's first point).

A *blob* represents a bounding rectangle on a connected set of similarly coloured pixels as well as some other properties of these pixels. Our language also operates on the properties and relationships between blobs to find objects. Both the definition and implementation of blobs in our system are a little different to that of other vision systems[1]. Blobs are often defined as a connected set of similarly coloured pixels and implemented with tree-based union-find operations. We have found that simply maintaining bounding rectangles with some other useful properties allows for faster performance with no penalty on blob identification.

Thus an *object* is a named entity in our language. It can be a blob, a line or a point or a set thereof. Objects are made externally available for post-processing by other modules on the robot. We may, for example, have an

<object> <id>PY_BEACON</id> <above> <touching proximity=0.25> <proportional error=0.25> <blob><colour>PINK</colour></blob> <blob><colour>YELLOW</colour></blob> </proportional> </touching> </above> </object>	entity(A, B, py_beacon) ⇔ above(A, B) ∧ touching(A, B, 1) ∧ proportional_to(A, B, 0.25) ∧ colour(A, pink) ∧ colour(B, yellow)

Fig. 1. A simple XOD representation and its logical equivalence for a pink/yellow beacon in RoboCup Four-Legged League

object called *ball* or we may also have an object called *field_edge* which is actually a set of vectorised lines representing the edge of the field.

3.1 Declarative Elements

The task of XOD is to represent objects of interest in a way that allows C++ code to be automatically generated for the task of locating the objects within an image if they are present. We use an XML based language because of its transportability and readability by humans and machines (but the language can be represented as logic clauses for even more readability, see illustrations and figures). The main construct in this language is the tag <object> which encapsulates the definition of an object. All objects have a property of a name indicated by the <id> tag. Many more features or properties are possible a la Object-Orientation or Frames.

To illustrate the language assume now that the object the vision system will attempt to recognise is to be identified via the use of colour (a common case in Robocup), and not by lines or edges. Then the object will be encapsulated (circumscribed/bounded) by a blob (if it is a single colour) or encapsulated by two or more overlapping blobs of different colours. Our language allows us to search for interesting blobs by both their properties (for example the <colour> tag) and the relationships between blobs (<above>, <in_front>, <touching>, etc) . For example, the XOD in Figure 1 could be used to locate a pink blob touching the top of and proportional to a yellow blob, useful if we were looking for the pink on yellow beacon in RoboCup Four-Legged League.

XOD also allows limitations on the blobs that we use based on the properties of the blob. For example if we were looking for a large orange blob that has at least 50% of the pixels within the bounding rectangle of the correct colour then we might write something like the XOD in Figure 2. Figure 2 also illustrates how the language will permit us to check relationships (or apply predicates) with other, previously defined objects such as the field edge. The truth value of these predicates are determined differently depending on the type of object being used. For example, the truth value of *below(A, FIELD_EDGE)* will need to be calculated differently depending whether *FIELD_EDGE* is an edge set or

<object> <id>FIELD_EDGE</id> <edge> <source>FIELD_BLOB</source> <colour>WHITE,GREEN</colour> <colour>YELLOW,GREEN</colour> <colour>BLUE,GREEN</colour> <vectorise>line</vectorise> </edge> </object>	entity(A, field_edge) ⇔ $a \in$A ∧ bounded_by(a, field_blob) ∧ [between_colours(a,white,green) ∨ between_colours(a,yellow,green) ∨ between_colours(a,blue,green)] ∧ straight_line(a)
<object> <id>CLOSE_BALL<id> <below> <blob> <colour>ORANGE</colour> <area op=gt>5000</area> <pixels op=gt>2500</pixels> </blob> <object>FIELD_EDGE</object> </below> </object>	entity(B, close_ball) ⇔ below(B, field_edge) ∧ colour(B, orange) ∧ greater_than(area(B), 5000) ∧ greater_than(pixels(B), 2500)

Fig. 2. A simple XOD representation and its logical equivalence for a large, primarily orange blob below the field edge line

<object> <id>FAR_BALL</id> <blob> <colour>ORANGE</colour> <pixels op=lt>500</pixels> </blob> <select> <area op=gt></area> </select> </object>	entity(A, far_ball) ⇔ colour(A, orange) ∧ less_than(pixels(A), 500) ∧ $\forall x$ greater_than(area(A), area(x)) ∧ colour(x, orange) ∧ less_than(pixels(x), 500)

Fig. 3. As an approximation to finding a far orange ball, here is an XOD definition finding the largest orange blob less than 500 pixels with its equivalent logic

a blob. In Figure 2 it is implemented as a set of edges to illustrate that objects do not necessarily need to be formed from blobs.

3.2 Imperative Elements

Our language goes beyond declarative statements to simplify using quantifiers. This is illustrated when more than one blob in an image matches the criteria in an object definition. In this situation we have the option to specify how to select the correct one or to save the list for post-processing. We do this via the <select> tag. We may choose either to keep the entire set or to select one object according to some criteria. Figure 3 illustrates the example where we want to choose the largest orange blob with less than 500 pixels.

We can choose to select by any of the attributes of the blob or by comparing to some other named entity. For example, it is possible to select the closest red blob to the object identified already to be the yellow goal. In the four legged league this could be useful in identifying the red goal keeper. In the absence of

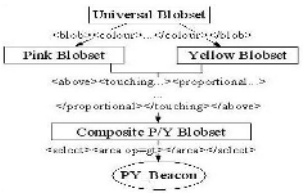

Fig. 4. The conceptual schematic pseudo-code of figure 1. The actual implementation has more efficient control flow

any select statement the default action is dependent on the type of object. If the object is a blob then the largest blob by area is selected however if the object is a point or an edge then the set is kept and the object remains a collection.

There are other imperative statements in the language used to modify or create objects rather than define or select them[3].

3.3 Implementation

Each XOD is converted to C++ code that performs the following three-step meta-procedure:

1. Build subsets of the universal blob set according to the properties specified in the <blob> section of the definition.
2. Apply predicates forming sets of composite blobs which represent intermediary and candidate objects.
3. Apply the operators according to the <select> statements.

Figure 4 illustrates the meta-code generated from the XOD example in Figure 1 (searching for a pink on yellow beacon). Firstly subsets containing respectively all the pink and yellow blobs are collated from the universal blobset. If there were other property restrictions defined on these blobs then they would also be checked at this stage. The second step is to compare every blob in list 1 against every blob in list 2 to see if the predicates defined by the relationship tags evaluate to true. If they do, then a new blob is created that bounds both of these blobs and it is added to the universal blobset as well as a new set containing those blobs generated in this stage. Properties are generated from the two source blobs and it is given a colour unique to this XOD. Finally the largest blob (by area) is selected from the set of composite blobs and named PY_BEACON because the XOD in Figure 4 has no <select> statement so the default action applies (select largest by area).

For efficiency, our implementation does not actually manipulate sets of blobs but rather indices to blobs in an array. All blobs (including composite blobs created by our procedure) are stored in a single, unchanging (with the exception of adding new blobs) array. We also perform several other important optimisations.

[3] <create>, <copy>, <set>

4 XOD Illustrations

4.1 Black and White Ball

One of the Robocup four-legged league challenges last year required teams to locate a black and white ball and then score a goal with it. The challenge was not performed successfully by any team in the competition. The XOD language allows us to specify a definition for the ball that requires only very minor alterations in the ball definition and the system will identify balls of non-uniform colour easily; in this case, a black and white ball. The XOD used to identify the black and white ball in Figure 5 differed only by three lines to the XOD used to locate an orange one.

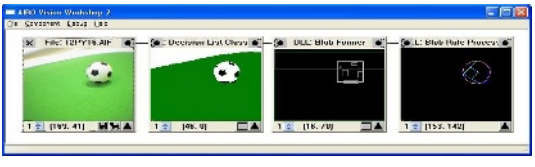

Fig. 5. The XOD used to locate the orange ball differs only by three lines to the XOD used to locate the the black and white ball

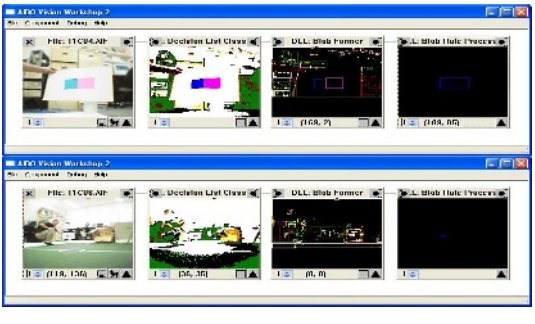

Fig. 6. AIBO Vision Workshop 2 running sideways beacon detection code. The XOD to find sideways beacons (flags) differs only in one word to the XOD used to locate normal beacons

4.2 Flag Instead of Beacon

One proposal to move Robocup more towards real soccer is to replace the navigation beacons with corner flags. The pictures in Figure 6 illustrate that our XOD system is capable of easily adapting to this change. Turning the beacons on the side requires only that the <above> relationship in the beacon definition be replaced with a <left_of> relationship.

5 Conclusion

XOD is an XML based language used to represent object descriptions in a code-independent way. This frees the vision system from requiring domain dependent knowledge to locate useful objects within an image. We have presented both the XOD language as well as an efficient implementation quick enough to be used on the Sony AIBO platform during a Robocup competition.

XOD enables us to quickly adapt to changing circumstances in the vision domain. For example, if the ball were changed to be a non-uniform colour (such as black and white) or the beacons were changed to be flags then the XOD definition of the objects can quickly be adapted to the new situation. The XOD also enables us to extend our vision processing application to domains other than the one it was originally designed for. We illustrated this by detecting an air-hockey puck. It is possible to use XOD to describe any object which can be easily identified using either colours or lines or by its relationship to other objects. This allows us to use XOD to identify many objects in widely different vision applications.

References

1. Bruce, Balch, Veloso: Fast and inexpensive color segmentation for interactive robots. In: International Conference on Intelligent Robots and Systems, IEEE Computer Society Press (2000)
2. Hornegger, J., Niemann, H.: Statistical learning, localization, and identification of objects. In: International conference on computer vision, IEEE Computer Society Press (1995) 914–919
3. Nayar, S., Murase, H., Nene, S.: Parametric appearance representation. In Nayar, S., Poggio, T., Nayar, P., eds.: Early Visual Learning. Oxford University Press (1996)
4. Draper, B.: Learning control strategies for object recognition. In Ikeuchi, Veloso, eds.: Symbolic Visual Learning. Oxford University Press (1996)
5. Lotzsch, M., Bach, J., Burkhard, H., Jungel, M.: Designing agent behaviour with the extensible agent behaviour specification language xabsl. In: 7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences), Lecture Notes in Artificial Intelligence, Padova, Italy, Springer (2004)
6. Draper, B., Ahlrichs, U., Paulus, D.: Adapting object recognition across domains: A demonstration. Lecture Notes in Computer Science **2095** (2001) 256