The Oxford Robot World Model

Bill Triggs and Stephen Cameron
Programming Research Group, Oxford University,
11 Keble Rd, Oxford OX1 3QD, U.K.
{bill,stephen}@robots.oxford.ac.uk

ABSTRACT

After reviewing the advantages of supplying a robot with a geometric model of its surroundings, we discuss the design of a modular object-oriented database to support such a model, which is being implemented as part of the Oxford Autonomous Guided Vehicle project.

1. Model Based Robotics

The Oxford Autonomous Guided Vehicle [1, 2] is a mobile robot truck or forklift designed to operate with a degree of autonomy in a structured and fairly well known factory-like environment. It has a number of advanced sensor systems for navigation, obstacle avoidance and pallet acquisition, and a corresponding set of task and path planning strategies, but at the heart of the software is a model-based representation of the vehicle and its environment, supported by a purpose-built geometric database.

There are several good reasons for providing a robot with such a world model:

(i) It provides an effective, easy to use, high level interface to the robot: task level behaviour such as 'pick up the pallet at x and carry it to y' is natural to a robot which 'understands' its surroundings.

(ii) It encourages a correspondently clean internal architecture in which independent vehicle subsystems such as sensing and data fusion, vehicle controlling, and path and task planning modules communicate with one another mostly via the world model. (Albeit there are usually a few time or safety critical mechanisms such as basic vehicle reflexes which must bypass the model). This modular, inherently parallelisable, model-based, data-driven architecture simplifies the introduction of new subsystems by providing a uniform inter-module interface; in particular it allows vehicle capabilities to be extended by the addition of more or less generic 'expertise modules' which need only interface with the relatively simple and vehicle-independent model database.

(iii) It decouples essentially robot-independent environmental information from the robot-specific software, allowing it to be handled by efficient specialist code such as the spatial database and shared between a number of robot and control processes. The result is an integrated model of the whole environment which is immediately extensible to large multi-robot systems and provides a solid foundation on which to build interacting-robot and factory-wide controllers.

NATO ASI Expert Systems and Robotics, Corfu, Aug 1990.

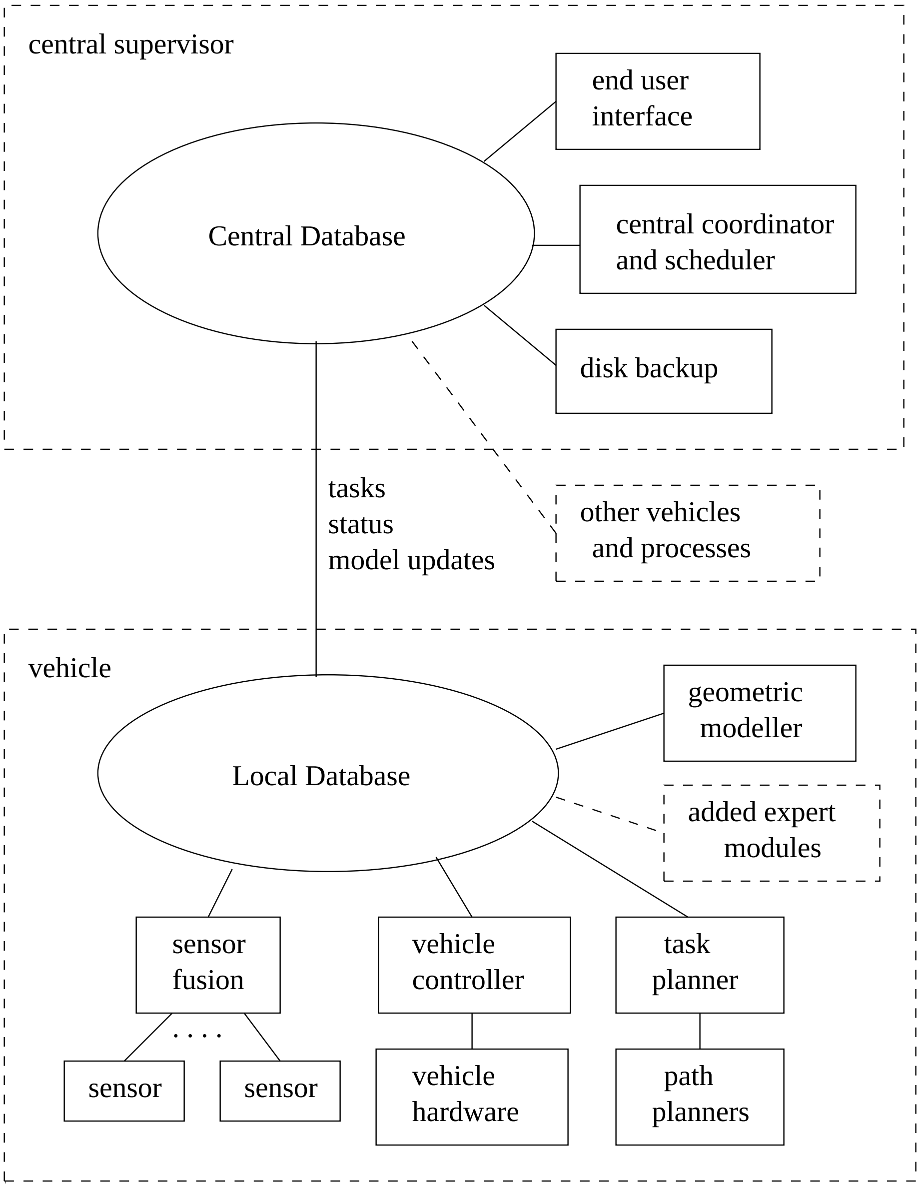

Figure 1: A modular architecture for a world-model based multi-robot system.

Fig. 1 illustrates the model-based architecture we are developing for the AGV project. Before discussing this in detail we shall briefly list some of the 'expert modules' being developed by other contributors to the project, which will eventually be interfaced to the world model:

(i) The Constructive Solid Geometry engine ROBMOD [3, 4], which was co-written by one of us, is being used for all of the solid modelling on the project. It provides a range of geometric expertise to its clients, including spatial bounds on objects for the indexing system, lists of visible edges for model-based vision, and very efficient three dimensional object interference detection for use by the path planning modules.

(ii) A range of model-based sensor systems are being developed, including a promising Kalman-filter-based sonar system [6], infra-red range and time-of-flight scanners, and single and multiple camera vision systems [7].

(iii) A number of different path planners are being evaluated, including generate-and-

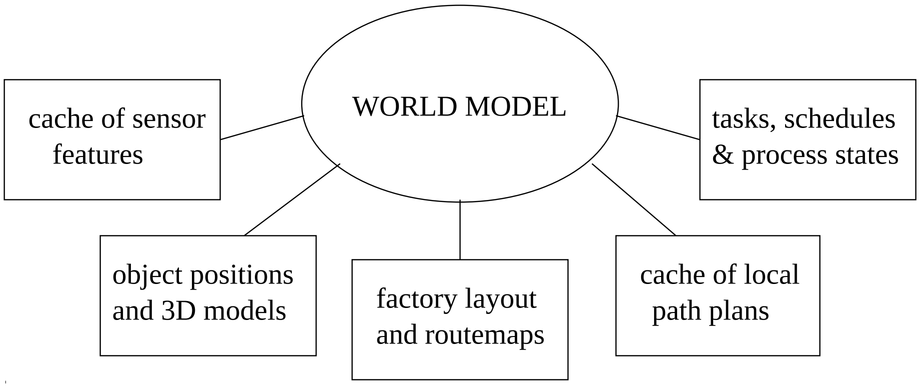

Figure 2: Types of information stored in the robot world model.

test, potential field and configuration space algorithms and several heuristics for common vehicle manoeuvres, with a view to incorporating some or all of them in a 'path planning expert' capable of choosing and invoking a suitable (combination of) method(s) for any given problem [5]. This will probably take the form of a rule base driving a battery of hard-coded planning algorithms.

(iv) An 'expert palletiser' is being developed to plan the task of clearing a region cluttered with pallets and obstacles. This requires the integration of the above path planner, a rule-based system enhanced with several graph-based heuristics to select a suitable order for clearing the pallets, and a docking manoeuvre planner to supervise the vehicle as it approaches and picks up pallets.

It is hoped to make all of these modules sufficiently flexible to be used with other vehicles or systems.

A useful robot world model must contain a wide variety of information, for example (see fig. 2):

(i) Representations of the physical layout of the environment and the objects within it such as walls, pillars, pallets and robot vehicles.

(ii) More detailed descriptions of objects such as three dimensional solid or boundary models.

(iii) Maps of features important to model-based sensor systems, such as visible edges and sonar-reflecting walls and corners.

(iv) A decomposition of the workspace into functional regions such as roadways, processing sites and stores, and a corresponding route-map graph for large scale route planning.

(v) Caches of reusable local or temporary data such as details of paths planned around local obstacles and partial feature maps of unidentified objects.

(vi) State, task, and scheduling information for the modelled robots and manufacturing processes.

The critical component in this model-based architecture is the database which underlies the world model: it must be able to store and access at least the above types of information; it must be particularly efficient for the common geometrically or spatially organised data; it must interface easily both with the end user and with a potentially wide range of robot subsystems; and it must be sufficiently flexible to deal with unexpected types of information and subsystem as neither of these are known perfectly in advance.

Moreover, even for a single robot system such as the AGV we feel that the model database should be capable of supporting a distributed multi-robot multi-process world model because:

(i) This allows the robot software to be simply partitioned into a number of independent modules, each with access to the database, as above.

(ii) Even a robot as autonomous as our AGV must exchange a considerable amount of task, state and environmental information with a land-based supervising computer over a (low-bandwidth radio) link, so that both vehicle and supervisor world models of some sort are required, together with a reasonably efficient data-exchange protocol. It seems sensible to use the same format for both model databases and organise the communications around database records, in other words to build a primitive distributed database system: this distribution may as well be built in to the database at the outset.

(iii) A robot designed around a multi-robot database is more easily adapted to an integrated multi-robot environment. Such integrated systems are interesting in their own right and are likely to be of very considerable practical importance in the future.

The support of such a multi-process system requires some sophistication of the database, in particular the consistency of multiple distributed copies of database records must be maintained in real-time, preferably without undue complication to either the user interface or the implementation.

2. The Oxford Robot Database: Design

Since we are not aware of any existing database which meets the above requirements, we are developing our own modular object-oriented database system specifically for robot modelling in a distributed environment.

In this system generic header structures known as 'Objects' are attached to all data items; for the most part the database uses only these header fields, however one such field describes the type of data contained in the Object and if necessary the database can manipulate this private data with the type-specific Object routines supplied with each type. The Object header is intended to provide a succinct description of the enclosed data suitable for use with a range of indexing systems, however it actually contains enough information to describe simple physical objects in its own right (see fig. 3). Fields are provided for an object name string and a unique system-wide object serial number, for a vector of user-defined binary 'object attributes' and a time-of-last-modification of the object, and most importantly for a spatial configuration (position and orientation) and a description of the spatial region occupied by the object. The main types of index currently supported correspond to these fields — in particular it is possible to access an object by name or serial number and by two or three dimensional spatial position — however new types of index are very easily added.

The spatial parts of the system are 'two and a half dimensional': all positions are three dimensional but coordinate transformations always preserve the vertical axis so that object orientation is given by a single angle. The spatial extent of an object is specified

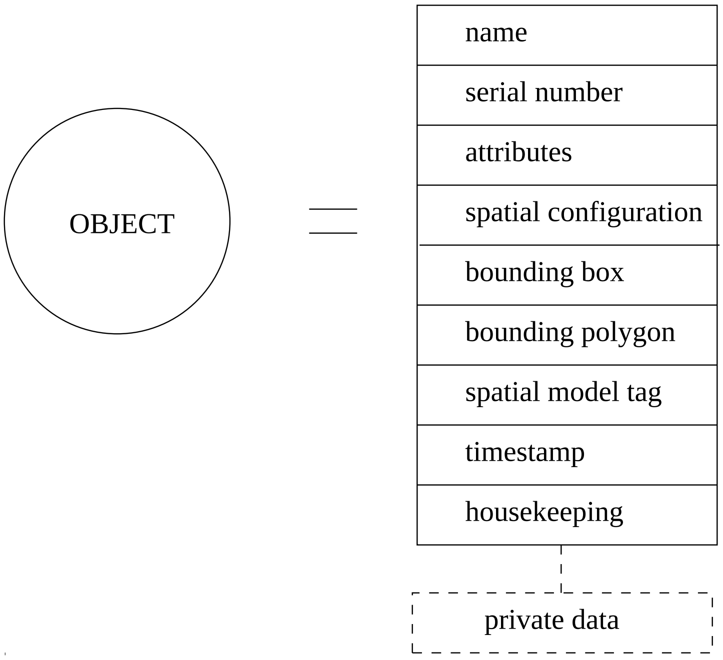

Figure 3: The Object header structure.

by a three dimensional bounding box (aligned with the current coordinate system) and an optional two dimensional inscribed polygon; these allow the database to perform fast but fairly crude object localisation and interference detection, and efficient spatial indexing. The database kernel deliberately does not have solid modelling expertise, since if full three dimensional shape descriptions or interference tests are required (for example for visible edge prediction or path planning in a cluttered region) a specialist geometric modeller can easily be run as a database subprocess. (There is a tag in the Object header for a modeller-based shape description). At present this role of 'system geometry expert' is filled by the Constructive Solid Geometry modeller ROBMOD [3, 4], however the modeller is not tightly coupled to the database and could easily be changed or omitted if required.

In operation, the database kernel attempts to maintain appropriate multiple copies of objects and all indices without user intervention, so that when an object is locally created, modified or destroyed the changes are automatically propagated to all database processes which have expressed an interest in objects of that class, and are reflected in all indices, local or remote, which contain or should contain the object. Thus, each client process can behave as if its own copy of an object was the only one and use a variety of indexing systems without explicitly having to maintain them. This process, illustrated in fig. 4, is achieved as follows:

All indices present a uniform interface to the user with primitives such as object insertion and deletion and a generalised search which performs a user-defined action on every object in the index which fits a user-supplied 'Object Template'. Object Templates are data structures designed to act as 'Object filters' and are used uniformly throughout the system to select interesting classes of objects, in particular every index has a private Template which determines which objects it will accept. Moreover, all indices within the system are actually Objects in their own right and may themselves be stored in system indices. This allows local indices to be maintained by keeping an index of indices and searching it for the indices which should be updated when a particular object is altered.

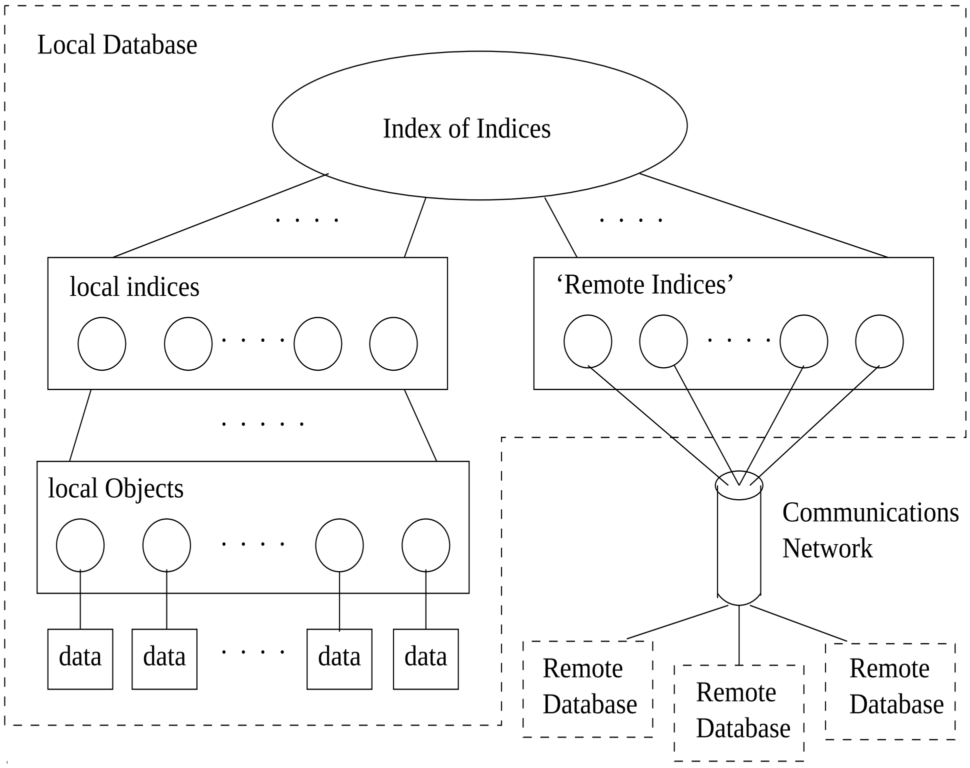

Figure 4: Architecture of the model database.

Similarly, a remote process with an interest in a class of objects lodges a 'Remote Index' in the local index of indices; this acts as a gateway to the remote process, transmitting appropriate classes of changes across a communications network to its parent. In both cases Object Templates are used for object selection and index search pruning.

We have not yet addressed the difficult problems of inconsistency or deadlock which arise when several processes simultaneously attempt to modify an object, however these are not expected to be severe in our domain as the few parts of the database which are frequently changed (such as robot states and positions) tend to be under the control of a single process.

3. The Oxford Robot Database: Implementation

At present the system is configured as a central database serving a number of sensor, controller and planning processes, each with embedded local database code. It is written in C and runs on (a network of) SUN workstations under UNIX, using SUN's Remote Procedure Call and External Data Representation mechanisms for network (INTERNET) communications in a machine-independent data format. The present implementation is only a prototype and the following changes are being considered: switching to an object-oriented language such as C++ would simplify the code considerably and allow a richer object hierarchy, although the loss of portability may be a problem; UNIX responds too slowly for realistic robot control and parts of the system will have to be ported to a different operating system, probably PARALLEL C or OCCAM as our vehicle controller is Transputer-based; and the Remote Procedure Call mechanism is not well adapted to our purposes and may have to be replaced with a lower level network protocol.

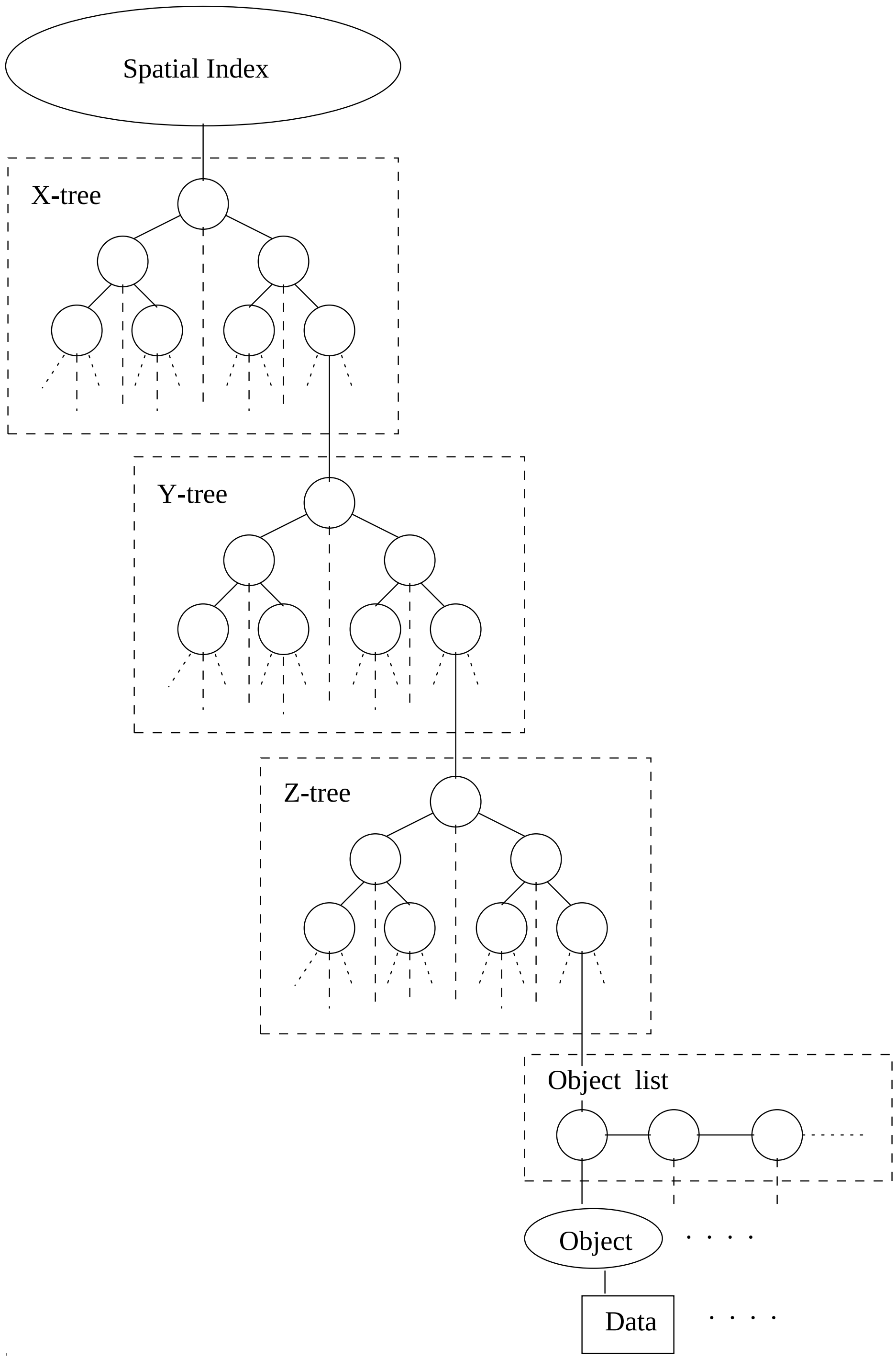

Figure 5: Spatial indices have a layered binary tree structure.

All of the basic types of indices are implemented using binary trees. The name and serial number indices are configured forms of a general purpose binary-tree index (with additional tree-balancing code) and the two and three dimensional spatial indices have a layered binary tree structure as follows:

Recall that spatial indices, being Objects, are provided with spatial configurations and bounding boxes; all indexing calculations are performed in index-configuration-aligned coordinates so that spatial indices with arbitrary spatial alignment are possible. For an object to be stored, its bounding box (in aligned coordinates) must overlap the aligned bounding box of the spatial index. If this is so, the index box is repeatedly bisected in the

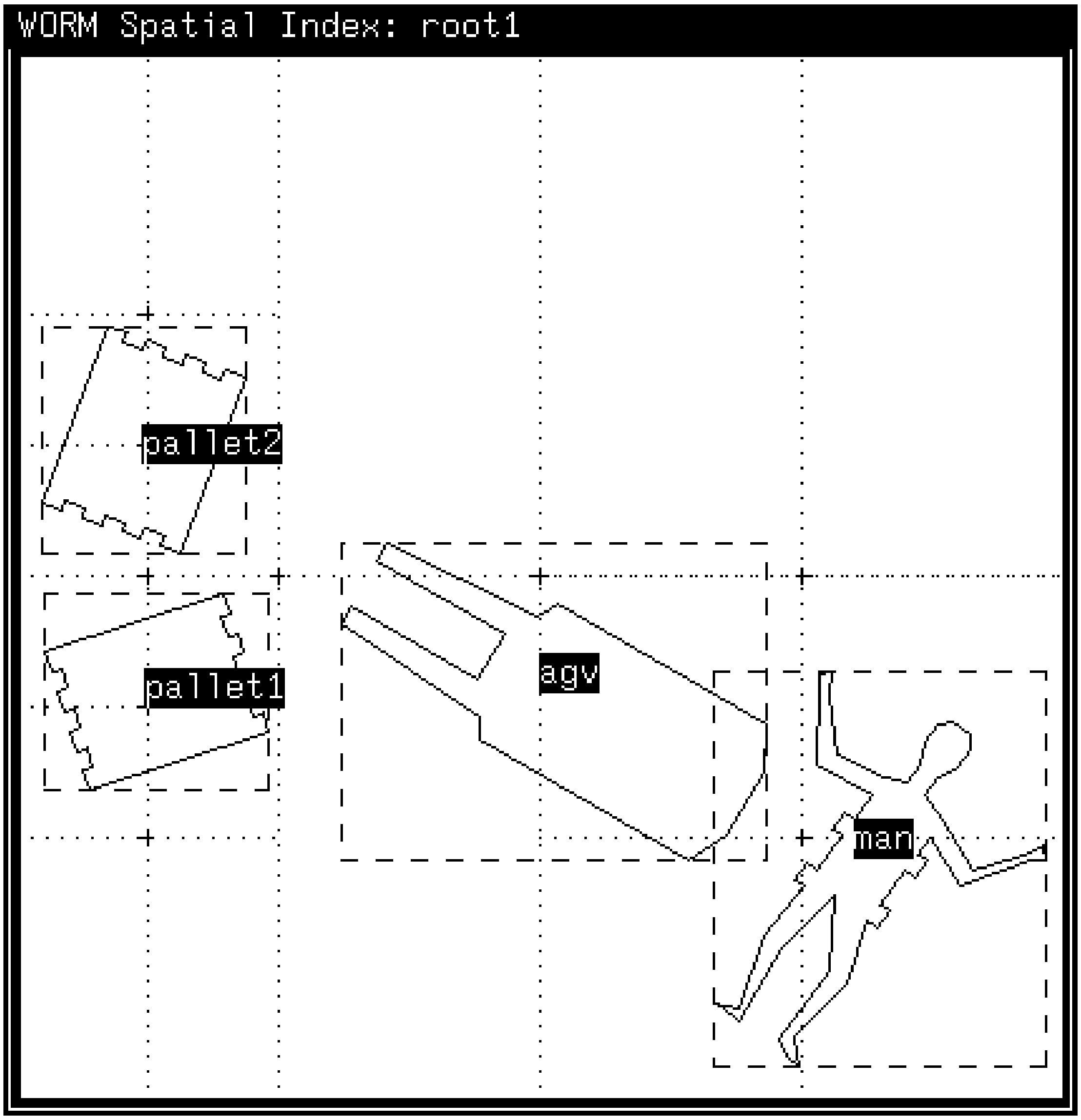

Figure 6: A two dimensional spatial index in action.

x-direction, the half containing the object being retained, until the object box is cut by a bisection rectangle; this rectangle is then bisected in the y-direction until the object box is cut by a bisection line; and the line is bisected in the z-direction until a bisection point falls within the object box; the object is then stored in a list of the objects associated with this bisection point. Thus, a three dimensional spatial index is a binary (x–) tree of binary (y–) trees of binary (z–) trees of lists of objects, as illustrated in fig. 5. Similarly, a two dimensional index is an x-tree of y-trees of object lists.

As is often the case with bisection-based spatial indices, each tree node is associated with a unique bounding box (the box which is bisected at the node), a unique spatial point (the centroid of the node box), and a unique set of three binary fractions (the coordinates of the node centre in the system normalising the index box to $[0,1] \times [0,1] \times [0,1]$), which can conveniently be used for node identification and tree traversal. Objects are stored at the node corresponding to the smallest node box containing the object and the first node centre to fall inside the object box.

Fig. 6 illustrates a two dimensional spatial index in operation. The polygons and bounding boxes of the objects are shown, and the dotted lines are node bisectors; thus, 'pallet1' is cut by the third layer of bisection on the x-axis (which also cuts 'pallet2') and the third layer bisection on the y-axis, and its fractional node coordinates are $(.001, .011)$.

To prevent very small or thin objects falling too deep in the tree a resolution cut off is usually applied, objects being stored in the smallest remaining node box containing them. It is possible to alter the order of the node bisections without affecting the depth of any object in the tree.

Note that the node boxes repeatedly overlap one another: this means that finding the objects overlapping a given box requires a search of all of the parents (containing nodes) and many of the children (contained nodes) of the box node; however the search is easily implemented by recursively eliminating nodes (and hence subtrees) which do *not* overlap the given box and this requires only a single coordinate comparison at each node.

There are of course many alternative ways of organising such a bisection-based spatial index; in particular quad-tree or oct-tree based representations [8] (in which the node box is divided into quadrants or octants at each level) might be thought to be the obvious choice. The chosen bin-tree system has a number of advantages over these:

(i) The bin-tree representation has node-boxes of arbitrarily high aspect ratio and is therefore extremely efficient at circumscribing long thin objects such as walls, corridors and rows of storage racks, provided they are aligned with the index axes. Such objects are obviously very common in factory-like environments. By comparison, the quad– and oct-tree structures (at least in their simplest forms) force all of the tree nodes to have the same aspect ratio as the original index-box and may therefore be rather poor at object localisation, particularly if the region being indexed is far from square.

(ii) The bin-tree representation leads to slightly simpler and more compact code and is trivial to adapt to indices of any dimensionality; in fact the two and three dimensional indices share the same code in our system.

(iii) Although it is true that quad– or oct-trees tend to be shallower than their binary equivalents, it does not follow that they are much faster to traverse or more compact. Our experience is that access time depends more on the total amount of bisection required than on the tree depth because most of the time is spent doing bisection arithmetic, and that (at least in the robot modelling domain) the higher branching ratio trees are actually less compact because most of the node pointers are never used. One caveat is that because the bin-tree system uses a larger number of smaller nodes it is comparatively more dependent on the efficiency of the memory allocator for nodes.

In practice this layered binary tree system has proved to be highly efficient at object localisation, simple, fast and reasonably compact, despite the fact that quite deep trees are generated.

4. Summary

Many robots would be significantly more flexible, powerful and easy to use if they were fitted with a geometric model of their surroundings, and this is particularly true of autonomous robot vehicles. When such a world model is supported by a properly designed database the further advantages of a modular, data-driven, database-centred system architecture with a clean, consistent user interface are available, and if this database is also able to support distributed processes the resulting system forms the basis of a powerful integrated multi-robot environment. A particular advantage of this architecture is the ease with which it can be extended as new expertise becomes available.

Running such a model-based, data-driven system makes considerable demands of the underlying database, however we feel that these will largely be met by the modular object-oriented robot database system which we are developing as part of the Oxford Autonomous Guided Vehicle project.

Acknowledgements

The AGV project is supported by the SERC ACME Directorate, grant GR/E62776, and is based on a vehicle provided by GEC-FAST. Our accommodation at the ASI was supported by NATO.

References

[1] Michael Brady, Stephen Cameron, Hugh Durrant-Whyte, Margaret Fleck, David Forsyth, Alison Noble, and Ian Page. Progress towards a system that can acquire pallets and clean warehouses. In *Fourth Int. Symp. Robotics Research*, pages 359–374, Santa Cruz, August 1987.

[2] S. A. Cameron. A geometric database for the Oxford autonomous guided vehicle. In B. Ravani, editor, *NATO ASI CAD Based Programming for Sensory Robots*, pages 511–526, Castelvecchio Pascoli, July 1988. Springer-Verlag. Vol. F-50.

[3] S. A. Cameron and J. C. Aylett. ROBMOD users guide. Software report, Department of Artificial Intelligence, University of Edinburgh (U.K.), 1987.

[4] Stephen Cameron and Jon Aylett. ROBMOD: A geometry engine for robotics. In *IEEE Int. Conf. Robotics and Automation*, pages 880–885, Philadelphia, April 1988.

[5] T. Hague and S. Cameron. Path planning for the Oxford AGV. In *Proc. IEEE Int. Workshop on Intelligent Motion Control*, Istanbul, August 1990.

[6] John J. Leonard and Hugh F. Durrant-White. A unified approach to mobile robot navigation. Technical report, Dept. Engineering Science, Oxford, UK, 1990.

[7] Ian Reid, Michael Brady, Alan McIvor, S. Marshall, I. B. Knight, and R. C. Rixon. Range vision and the Oxford AGV. In *Image Processing 89*, pages 51–72, Wembley, 1989. Online publications.

[8] Hanan Samet. The quadtree and related hierachial data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.