Appendix H

Software systems

H.1 Introduction

Several software systems were developed by the author to support the work described in this thesis. They are written in C++ and Perl, and run under the Linux/X-Windows environment. The software modules and files used to support intelligent robot control simulation are shown in figure H.1. Briefly, they are:

- `biped.dnd` describes the dynamic system that is the robot's "brain", in DND (dynamic network description) language (see Appendix J).
- `dnd2exe`, a program which compiles DND into C source code, producing the file `biped.c` (it is part of the `Dyson` system described in Appendix J).
- The system compiler `gcc` compiles `biped.c` into `biped.so`, a shared library object.
- `biped.cfg` describes the mechanical structure of the robot, and also contains some simulation parameters. This file is used by the RoboDyn library.
- `CyberSim` perform the actual simulation.
- `cybersim-results.rsdf` is a binary data file that contains the state (position and velocity of all the mechanical elements) at each time step.
- `dsdata.rsdf` is a binary data file that contains the internal state of the "brain" dynamic system at each time step.
- `CyberView` is an interactive rendering program which allows the user to move through a virtual 3D environment and observe the operation of the robot over time.
- `DSVIEW` is an interactive application which allows the user to graphically view the dynamic system variables over time.
- `RoboDyn` is a library which performs the dynamical mechanical simulation of the robot.
- `FoxN` is a library which implements the FOX algorithm.
- `R3D` is a library which performs 3D rendering.

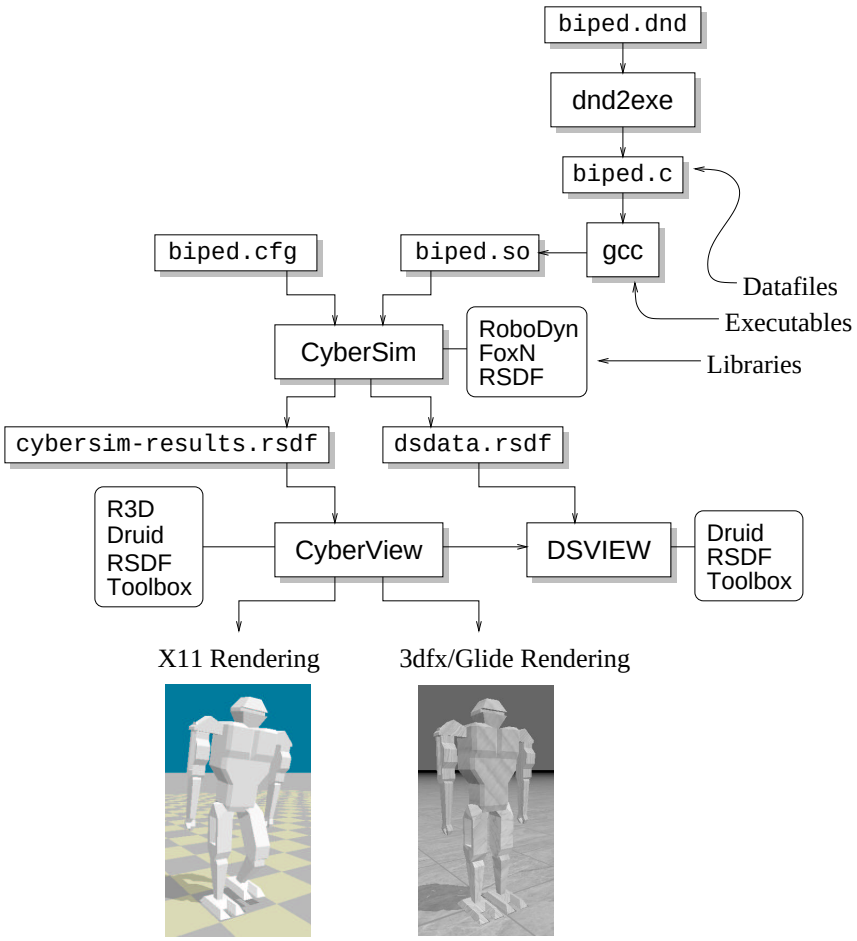

Figure H.1: *The software modules and files used to support robot intelligent control simulation.*

- `Druid` is a GUI class library and widget set.
- `RSDF` is a library which reads and writes files in the RSDF format.
- `Toolbox` is a library which implements garbage collection, runtime type identification (RTTI), exception handling, interfaces, and hash tables.

The major systems are briefly described in this appendix, except for `dnd2exe` (which is described in Appendix J) and `RoboDyn` (which is described in Appendix I)).

H.2 CyberSim

CyberSim performs 3D mechanical simulation of a robot together with its intelligent controller. It brings together the `Dyson`, `FoxN`, `RoboDyn` and `RSDF` libraries. It takes two files as input: a `RoboDyn` configuration file which describes the mechanical structure of the robot, and an object file which is dynamically linked to CyberSim to implement control functions. The configuration file also specifies the simulation time step size, the number of iterations to perform and the integration method (Euler, midpoint or Runge-Kutta fourth order). It generates data files in RSDF format containing the mechanical and controller system states at every time step.

H.3 CyberView and DSVIEW

CyberView (figure H.2) is an interactive rendering program which allows the user to move through a virtual 3D environment and observe the operation of mechanisms simulated with CyberSim. It displays the objects stored in CyberSim data files. The custom graphics engine (the `R3D` library) supports flat shaded rendering to an X-terminal or texture mapped real-time rendering to a 3dfx/Voodoo graphics accelerator card.

CyberView allows an external data display (EDD) program to be invoked and synchronized with CyberView's currently displayed time step. One such program is DSVIEW (figure H.3) which displays the internal variables of the dynamic system that controls the mechanical model (see Appendix J).

CyberView's main window is shown in figure H.2. The user can move around the virtual environment by clicking and dragging in the 3D display area (the mouse button that is pressed determines the set of axes that are moved along). The simulation movie can be played using video recorder style buttons. The menu options allow the user to open and close data files, start an EDD program, save a single frame or a sequence of frames as PPM graphics files, change the lighting angle, set and jump to up to 10 camera positions, render in wire-frame or solid mode, and set the shadow and z-buffer parameters.

H.4 R3D

The `R3D` library used by CyberView performs the following tasks: representation of the drawing primitives (3D polyhedral objects and infinite planes), geometric transformation between object space and screen space, clipping of drawing primitives to the viewing frustrum, lighting calculations, z-buffering for X11 rendering, texture mapping computations and shadow mapping.

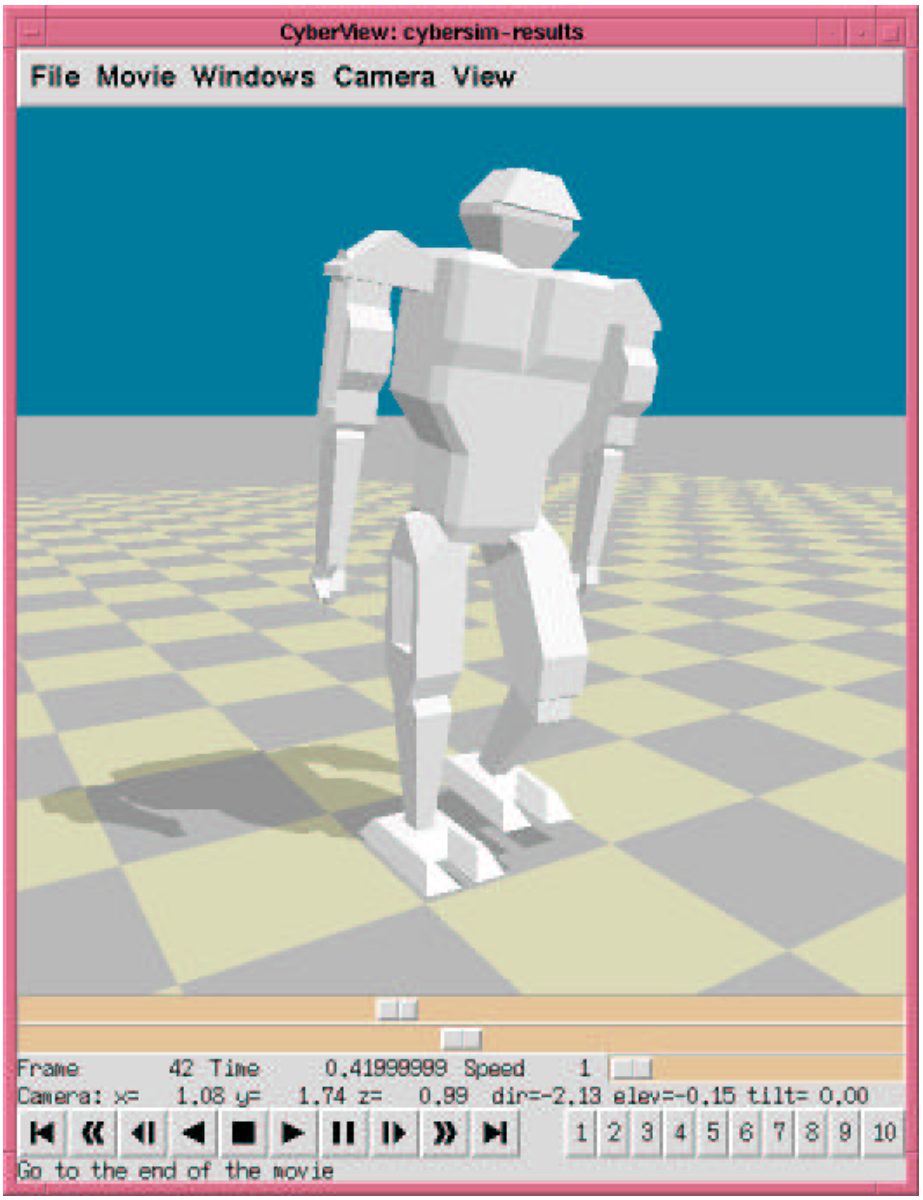

Figure H.2: *The CyberView main window.*

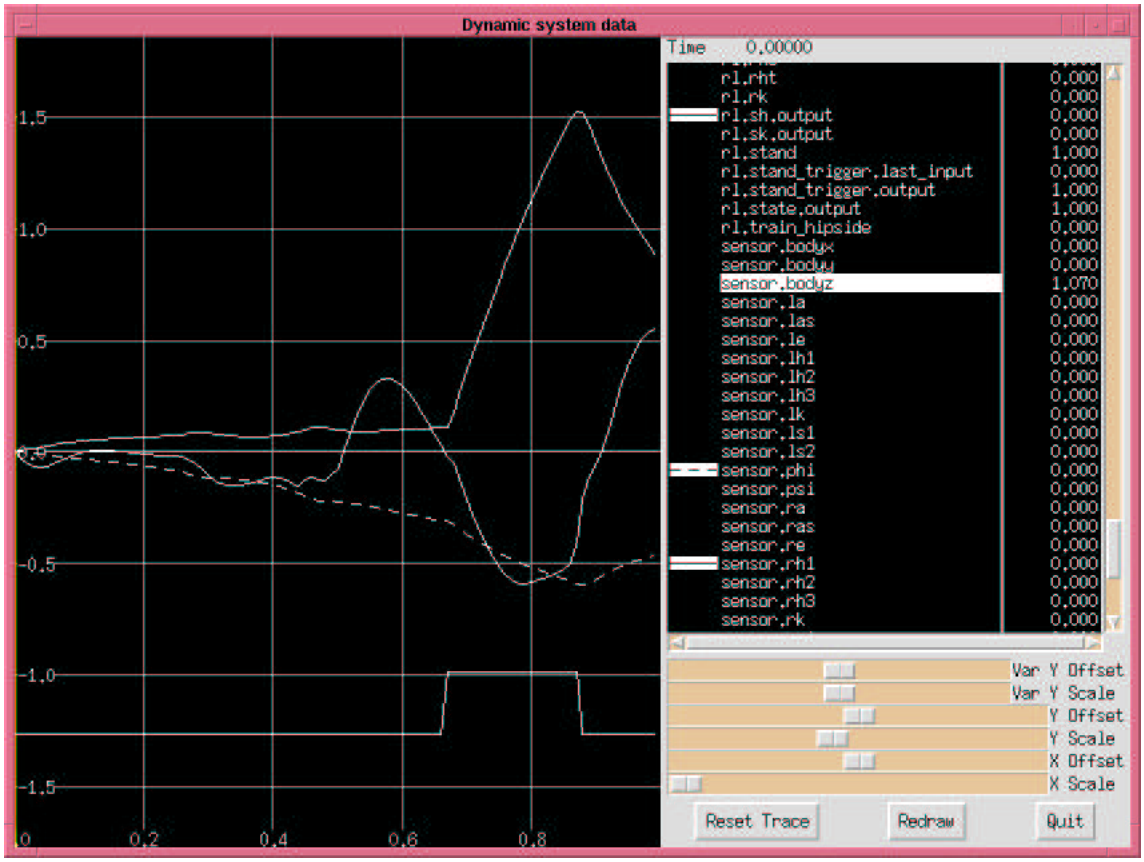

Figure H.3: *DSVIEW: The dynamic system viewer main window.*

H.4.1 The multi-resolution Z-buffer algorithm

When `R3D` renders to an X11 window, it draws flat-shaded polygons with hidden surface removal. A very simple hidden surface algorithm is to give each polygon a depth map and use the z-buffer algorithm [35].

However, with the normal z-buffer algorithm every pixel in every polygon must have its depth tested against the z-buffer. In software this is very slow compared to the speed at which an accelerated graphics card can render pixels. Note that anything more complicated than just flat-shaded polygons needs to be rendered (like Gouraud shading or texture mapping) then the z-buffer speed problem is insignificant compared to the extra processing required.

The `R3D` library uses a "multi-resolution z-buffer" algorithm so that it only has to do the full resolution (or fine resolution) z-buffer process on small blocks of the image, which mainly correspond to places where more than two polygons intersect. Large areas of constant color are drawn quickly without z-buffer computations.

Say the image size is 512×512 pixels. First the polygons are rendered on to a coarse grid of, say, 32×32 cells. Cells are classified as either:

1. Unused.
2. Completely filled by one polygon.
3. Filled by more than one polygon.

Type-1 cells are ignored. Type-2 cells are immediately filled with the appropriate color. Type-3 cells get the full z-buffer treatment. The majority of pixels in the image are hopefully going to be in type-1 or type-2 cells, so they can be processed quickly. Type-3 cells (where full z-buffering occurs) are only needed in the interior edges of an image. As the image complexity increases (i.e. more edges) then the effectiveness of this approach will be decreased. It will have the greatest improvement over z-buffer in cases where there are relatively few, large polygons in the image. An image that has a large number of small polygons (e.g. a gridded object) may not be rendered very efficiently compared to straight z-buffer.

H.5 FoxN

The `FoxN` library implements the FOX algorithm described in Chapter 5. It defines several C++ classes:

- `TraceN`, which performs the eligibility trace computations.
- `FoxN`, which performs the CMAC mapping and the main part of the FOX algorithm.
- `GetAtoi`, which allows fast table-based matrix exponentiation (it is used to compute the A^i values).
- `HPFloat`, a high precision floating point number class, for performing experiments on FOX systems that suffer from the small-eigenvalue precision problem (see Appendix F).

H.6 RSDF

RSDF is an acronym for Realm Structured Data Format. It is a file format used to store CyberSim simulation results. The file format is specially constructed to get the following features:

- Flexibility: the format of the stored data is a nested series of structures and arrays. The structure/array definitions are specified in the file header.
- Reading speed: all data is properly aligned so it is easy to `mmap()` the file into memory and access it directly.
- Accessing ease: When `mmap()`ed, nested arrays can be accessed with a simple indexing calculation, e.g. the address of `x[i].y[j].z[k]` is `k1*i+k2*j+k3*k+k4`.

H.7 Druid

Druid is a C++ GUI widget library for the X-windows system. Druid is fast and small, and supports rapid application development with a complete set of GUI widgets. It was written to explore OOP design strategies, and is used extensively in the other systems. These are the classes provided by Druid:

`CComponent`	For reference counting, RTTI, interfaces, exceptions.
`CTextArray`	Container for text arrays.
`ZColor`	A read-only sharable color in the default colormap.
`ZDrawable`	Represents anythings that can be drawn into.
`ZPixmap`	An X11 Pixmap.
`ZWindow`	An X11 window.
`ZFont`	A font and associated metric information.
`Zgc`	Represents an X11 graphics context.
`ZPalette`	Standard palette.
`ZPlugin`	Abstract module to give a window extra behaviors.
`ZAutoHelp`	Automatic status-bar help.
`ZFocus`	Participate in keyboard focusing.
`ZPacker`	Geometry-packs sub-windows.
`ZPackInfo`	Information packet for `ZPacker`.
`ZScroller`	Scrollable sub-window.
`ZTopFocuser`	Top-level focus window.
`ZWidget`	A widget is a window with palette and font.
`ZButton`	A pushbutton...
`ZPixmapButton`	...that displays an image.
`ZTextButton`	...that displays text.
`ZFrame`	A 3D frame.
`ZLabel`	A static text label.
`ZLineEdit`	An editable text box.
`ZMenuBar`	A menu bar.
`ZScrollbar`	A scrollbar.
`ZSlider`	A position-slider.
`ZStatusLine`	A status line.
`ZTextList`	A list of text items.
`ZWidgetFont`	A plug-in widget font.
`ZPrivateColor`	A private read/write color cell.
`(XSetWindowAttributes)`	
`ZSetWindowAttributes`	Allows window attributes to be set.

`ZMenuBarItem`	A menu bar item.
`ZMenuItem`	A popup menu item.