Vision Servers and Their Clients

James J. Little

Technical Report 94-19

October 1994

Laboratory for Computational Intelligence
Department of Computer Science
University of British Columbia
Vancouver, British Columbia, CANADA V6T 1Z4
email: little@cs.ubc.ca

Abstract

Robotic applications impose hard real-time demands on their vision components. To accommodate the real-time constraints, the visual component of robotic systems are often simplified by narrowing the scope of the vision system for a particular task. Another option is to build a generalized vision (sensor) processor and provides multiple interfaces, of differing scales and content, to other modules in the robot. Both options can be implemented in many ways, depending on computational resources.

The tradeoffs among these alternatives become clear when we study the vision process as a server whose clients request information about the world. We model the interface on client-server relations in user interfaces and operating systems. We examine the relation of this model to robot and vision sensor architecture and explore its application to a variety of vision sensor implementations. [1]

[1]This research was supported by the Natural Sciences and Engineering Research Council of Canada and the Networks of Centres of Excellence Institute for Robotics and Intelligent Systems, Project A-1.

1 Introduction

The goal of real-time performance conflicts with the computational intensity of most vision tasks. Several constraints on architecture arise from real-time operation, including: I/O signal bandwidth, time constants, feedback loop constants (reaction times), and dynamic environments.

Vision systems often simplify to respond to real-time constraints—if there is little time to do something, do very little. Many robotic systems employ specialized sensors designed for particular tasks so that the connection between sensor and action can be direct and fast[FB89]. Each path in the robot, connecting the world to sensors, computation, and effectors finally back to the world, is a *behavior*; the enormously influential subsumption theory[Bro87] describes how to connect each sensor-to-actuator *loop* with inhibition and suppression contacts. These loops avoid a centralized modeling and planning facility and therefore can run in real-time.

Reactive robots built on the model of direct sensing-actuation connections have proliferation, but their limitations have become obvious. The effort to return to robots that plan and react has led to the reactive/deliberative debate. Connell[Con92], for example, has developed a three-level model of robot architecture: the lowest is a Servo loop, running in real-time; the middle is the Subsumption level, and the highest is the Symbolic level. Only the lowest level forms a sensing-actuation loop with the world, but the other two levels communicate with the Servo level hierarchically.

Likewise the RCS model of Albus[Alb92] (widely accepted in the robotics community) advocates a collection of loops organized in a matrix. Each loop is a level; each level has successively coarser time sampling and a correspondingly lengthier time horizon. Within each loop are sensing, planning and control components. Communication occurs across a loop, and between levels among corresponding components. For example, the sensing component at the 10 ms level talks with the the sensing component at the 100 ms level. One advantage of the RCS formulation is the prominent role it gives to time delays, sampling intervals, and planning horizons. The low-level layers usually operate with fixed-schedule of computation, while the higher layers are event-driven.

These models describe logical architectures. Resource and timing constraints shape their realization on physical architectures. Large computational tasks argue for large monolithic machines, but the variety of tasks argues for heterogeneous machines. Bandwidth requirements argues for close coupling, while real-time distributed systems argues specialized communication channels. Many variations of architectures have explored all these dimensions.

Increasingly vision sensors serve multiple purposes within a robotic architecture. A vision sensor provides data for a servo control loop as well as data for recognition, manipulation and localization. The sensor is a critical element of any control loop, even if the robot treats the world as a source of symbolic predicates[Fir92]. The controller must sense its progress during actions. Control raises the issue of delays: systems must consider latency as well as throughput. The robot cannot be blind during actions. It must have an alert mode, so that it senses during movement. Coverage by the sensor system must be broad to monitor the environment for threats, albeit as simple as an obstacle during locomotion.

Matthies and Grandjean[MG93] develop a methodology for integrating sensing and action. The systems senses (stereo ranging) a portion of the scene, sufficient to act within its sensed region. They identify the important constraint of the robot's reaction time in the design of the sensing component.

1.1 Vision Operating Systems

The right model for the design of a vision system can be taken from the design of operating systems. What services must a vision operating system provide to let processes operate continuously, cooperatively, and transparently? Early vision processing can be implemented, because its regularity, in specialized parallel processors such as mesh and pipeline processors. Later processing stages are data dependent and complex and must be implemented on more general processors. On general processors, early vision forms the bulk of the computational burden.

We structure real-time vision systems as processes that operate on data streams. Each process typically operates on an image, then optionally subdivides an image into smaller subimages and sends the image(s) on to later processing stages. The later stages can collect several image streams, representing subimages of an original image, to form aggregates.

The vision process that produced the visual data is the *server*. Later stages that use the data are its *clients*. A client can play several roles: it can just pass the data on, either unchanged or processed; more generally it consumes the data and produces other data.

Coordinating servers and clients can be problematic when they operate at different speeds. To solve the problems of implementing non-trivial vision processing on realistic machines, a designer must confront the realities of *multi-rate* processing: due to technology mismatch between early processing engines and more limited intermediate and high level processing capability, as well as limited communication bandwidth, the rate at which components in the system produce and consume data vary enormously.

One example of a service implemented in a vision OS is A *smart buffer*[LK93] that performs intermediate processing on arriving data to connect two processes at different speeds, providing current data without burdening the receptor process with details of how it is buffered. Synchronization is an important issue, especially when the system serves control loops. Interrupts and exception-handling are already part of robotic systems, including those based on vision.[de93]

In many respects, the "whiteboard" concept supported by CODGER (COmmunications Database with GEometric Reasoning) in CMU NAVLAB [SST86] provides an abstraction for vision services and clients. CODGER supports data flow between parallel modules using a central blackboard database. Unlike CODGER, a smart buffer operates essentially to collect data. Like CODGER, the buffer handles synchronization between the producer and consumer. Information sources are servers, and information consumers are clients. Unlike the blackboard, because of the high cost of delays, we want instead to connect services with clients via direct paths, as in the dataflow model.

A system that serves multiple purposes will likely have several sensing modalities that can be fused. Integrating multiple modalities leads to robustness; independent sources can fuse data for verification. Data flows must nevertheless remain accessible as data sources to other tasks, much like a multi-ported memory. Data sources connect both with clients and constraints. A centralized implementation, moreover, can support a decentralized model by synthesizing logical sensors[de93] from existing inputs. For example, binocular stereo data can serve to synthesize a spot range sensor and vice versa.

2 Examples

The following examples demonstrate some possible configurations of sensory systems and control systems in robotic architectures. The systems have been designed with regard to several architectural constraints on real-time operation including I/O signal bandwidth, feedback loop constants, and response to dynamic environments.

The three systems include various vision services. The tracker system uses a motion/stereo data source for a target selection and tracking client. The Dynamo soccer-playing testbed has a real-time color tracker to supply positions of several soccer players to a distributed implementation of soccer playing controllers. Finally, the real-time localization system uses on-board video with off-board computation to determine position and orientation at 10 Hz.

2.1 A Multi-rate Motion Tracker

The motion tracker[LK93] demonstrates how to build a tracker that has several processes running at different rates in near real-time. The tracker follows a moving object, with no knowledge of its target, based on dense optical flow input.

A simplified tracker can observe the world only when its head is static. The "stop-and-look" mechanism excludes imaging while moving. Such a tracker system cannot detect change while analyzing data or moving its head. Given a sufficiently high-speed eye-head platform, such as Yorick [SMV+93], one can operate on pairs of frames, in a "stop-and-look" fashion.

The UBC Vision Engine[LBKL91] exemplifies distributed, heterogeneous systems that have multi-rate processes. The Engine is general-purpose and supports all levels of vision. The system consists of multiple architectures: pipelined (a Datacube MaxVideo200) image processor and a MIMD multicomputer (20 T800 2MB Transputers, connected via a crossbar), connected by a bidirectional video-rate interface. The Transputer system controls an eye/head platform for vergence, pan and tilt.

Our tracker continuously monitors its environment, overlapping interpretation and sensing. It identifies the largest moving object as the target. Since the tracker gathers visual data during head movements and identifies the target solely on motion and stereo cues, it must compensate for the apparent motion caused by head movement. It estimates the image motion caused by camera motion and *cancels* the apparent motion in the accumulated optical flow. It is then simple to find the target since the moving object "pops out" relative to a static background.

The tracker has multiple stages: correlation motion on the Datacube, optical flow accumulation, target selection and eye-head control. The first two stages comprise the Perception component, distributed over two different machines. The next stage is the Reasoning component, which analyses the Perception output to determine the target. The final stage translates the target location in image coordinates into controls for the eye-head. Figure 1 depicts the data flow in the system.

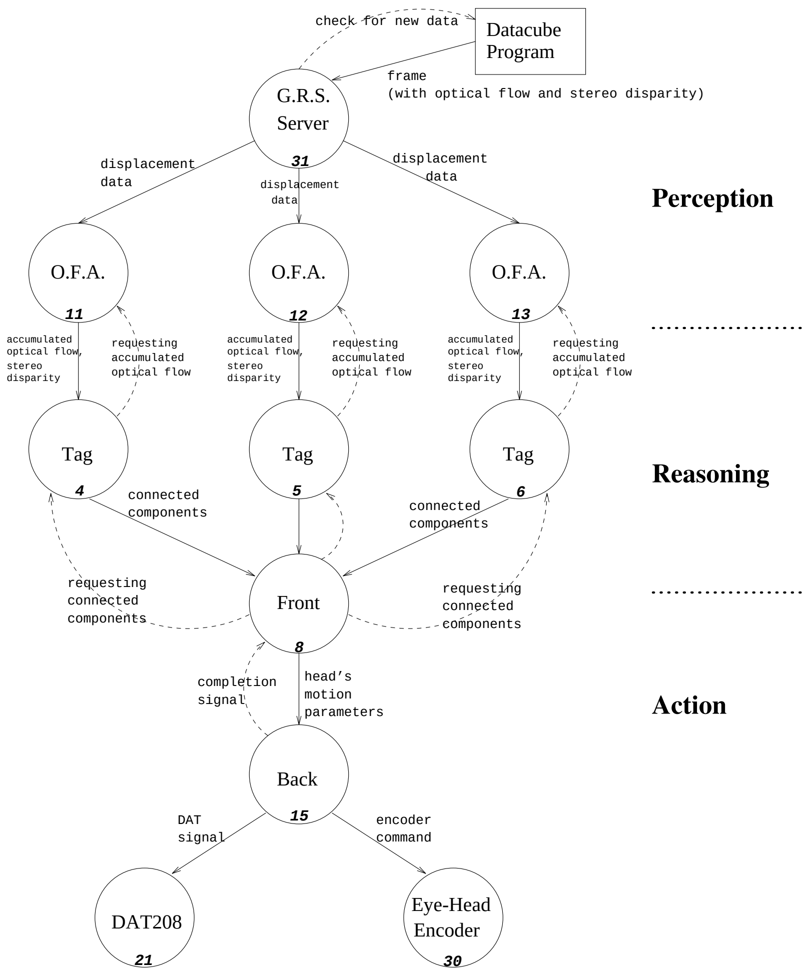

Figure 1: Software Components and Data Flow

A *smart buffer* process[LK93] is responsible solely for receiving the motion data stream. The smart buffer contains an *active monitor* that waits for data to arrive, and keeps track of the data until it has been properly stored for later retrieval by the reasoning system. The monitor is "active" since it asynchronously accepts motion data whenever it is available.

The active monitor plays an important role in synchronizing communication and uniting the different modules. Optical flow and stereo disparity images are pumped out continuously regardless of whether the reasoning system is prepared to process the data. The monitor must execute at a rate *no slower* than the rate the displacement measures are being pumped out; otherwise, the loss of any data frame contributes to the inaccuracies of the overall system. This is especially critical since the system accumulates displacements over several frames. Figure 2 shows the structural relation between the active monitor and other vision processes.

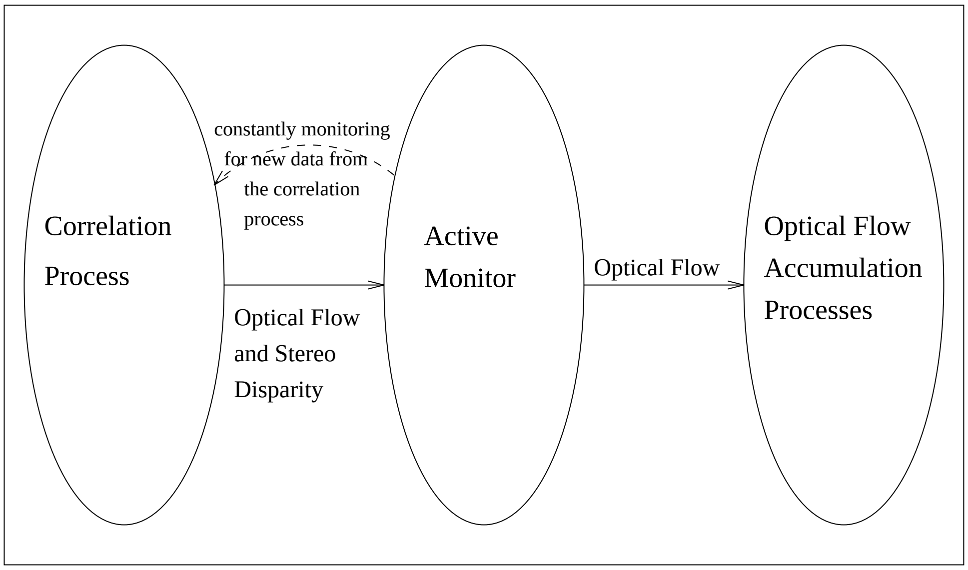

Figure 2: Perception System

The Reasoning stage finds connected components in 300 to 500ms; this is the principal delay in the system, necessitating the multi-rate interface. The Reasoning stage sees the smart buffer as a data object that contains the optical flow. The active monitor sees the smart buffer as its client. The full cycle, optical flow computation and accumulation, flow cancellation and target finding requires 800ms, mostly due to component labeling.

An important constraint required for cancellation is that the eye-head complete its commanded movement before the reasoning system requests optical flow data from the smart buffer. Incorrect cancellation occurs when the motion is not complete. Thus, the stage synchronize using message-passing, when the head controller completes the motion. Currently the system is able to track a person moving at a normal walking pace 2 meters from the cameras.

The system has two clients: the target process needs accumulated motion data, and a filtering process needs current depth data from stereo. Filtering

Figure 3: Dynamites

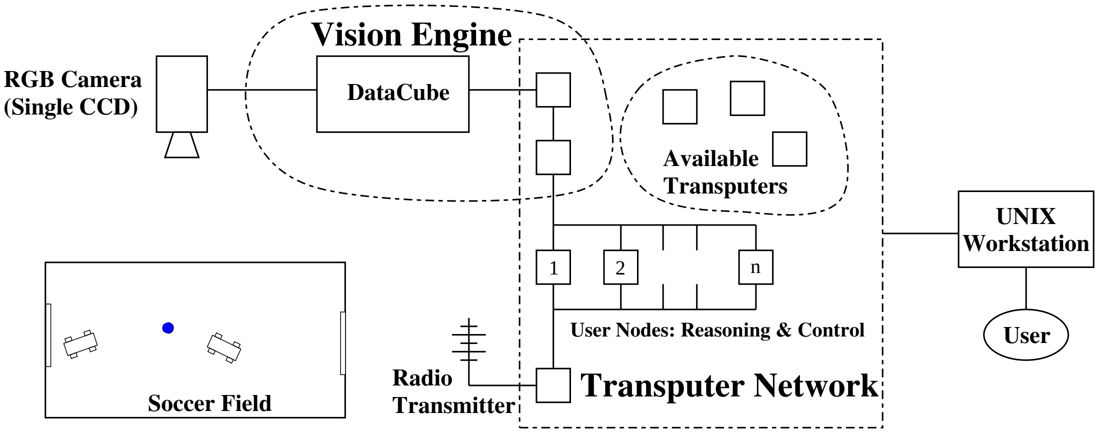

Figure 4: Dynamo Architecture

selects a target in a specific range of depths. To connect a third client, for example, one to use stereo data cues or motion cues to detect looming, the system need only arrange that the stereo and motion buffers provide read-only access to a daemon that detects looming.

2.2 Color Tracking for Localization

The Dynamo testbed is a collection of independently controlled mobile robot vehicles that play soccer [BKL+93]. The system demonstrates offboard vision processing and distributed processing. The vision component was originally prototyped on the Datacube MaxVideo200. Currently the system is realized as a simple custom hardware to process RGB signals, followed by run-length encoding and centroid calculation on Transputers. A single off-board camera sensor communicates its signals to the centralized sensor processor. The sensor processor provides positional information to the control processes for each competing soccer player at 60Hz (once per image field) with a lag of at most 5 ms after the end of field. The structure of the full system is shown in Fig. 4.

The Dynamo system has been used to explore novel reactive strategies for control[Sah94] as well as the testbed for ideas on control, specification, and reasoning about real-time systems[ZM92].

2.3 Real-time Localization

ROLL (Real-time Onboard Localization with Landmarks) identifies its position in real-time, using passive visual localization of a single landmark[Bre94]. ROLL is implemented on a Real-World Interface platform (see Fig. 5), named Spinoza, with offboard video processing sent via FM narrowcast (antenna at left). Real-time processing occurs on a TI C40 system. The landmark (Fig. 6) contains a visual bar code as well as a structure that contains disks whose centroids can be rapidly and accurately computed. Their relative positions plus the known dimensions of the target allow ROLL to compute its own position and orientation at 10Hz. ROLL controls the RWI base via a spread-spectrum modem (at right in Fig. 5(a)).

The system is distributed over the C40s and a Sparcstation, which communicates with the RWI. Robust localization can use the platform's odometry, but there is significant latency between the odometry and the vision server on the C40s. Prediction by Kalman filtering must compensate for the varying temporal shift between the measurements.

3 Discussion

Vision systems need tasks to keep them honest and nothing is more honest than real-time deadlines, aptly named. Recently robot systems have used vision to sense the world at many scales, including within real-time control loops.

We have examined a collection of architectures that implement a variety of vision processes, from simple to complex. Such systems must permit overlapping sensing and action, with interfaces that allow transparent access to data at many

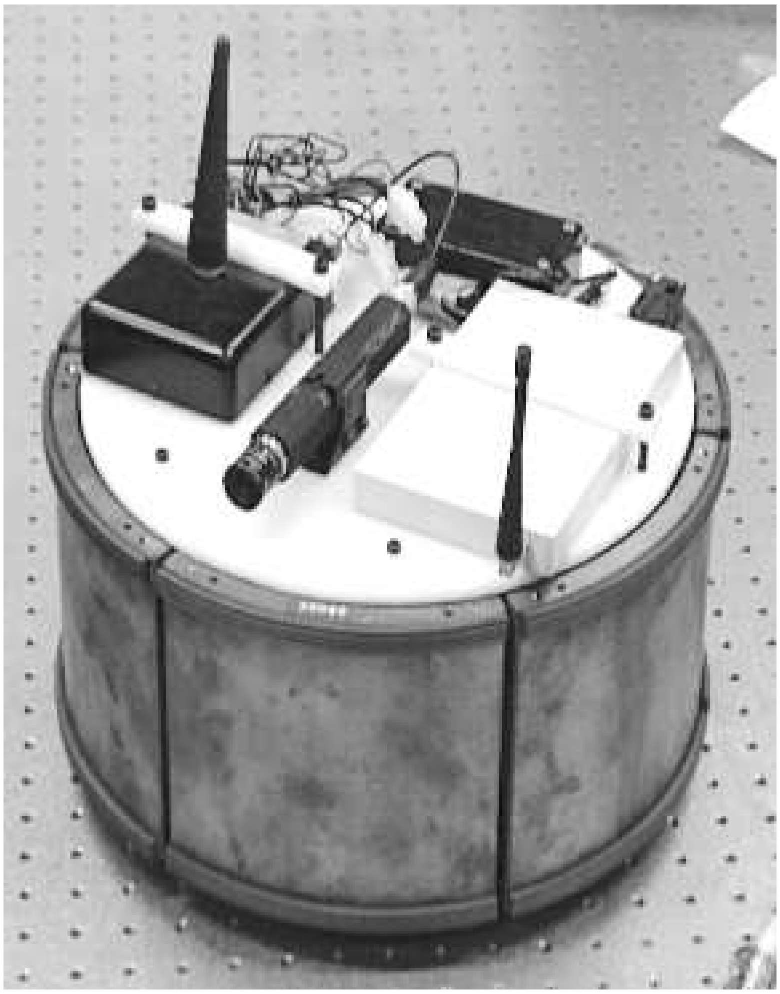

Figure 5: Spinoza

stages of processing.

Vision servers and clients can be implemented in a variety of configurations. Nevertheless, synchronization, communication and other issues from operating systems become critical in real-time operation. Centralized information sources/providers allow integrated processing/fusion and should nevertheless be implemented to permit simple access to separate elements of the data and at different stages.

One type of vision server, a smart buffer, exemplifies an interface between components of a vision system where substantial early vision computation occurs at high rates and later stages operate at slower cycles. The buffer serves to synthesize a virtual data source that can be interrogated at any time by mid-

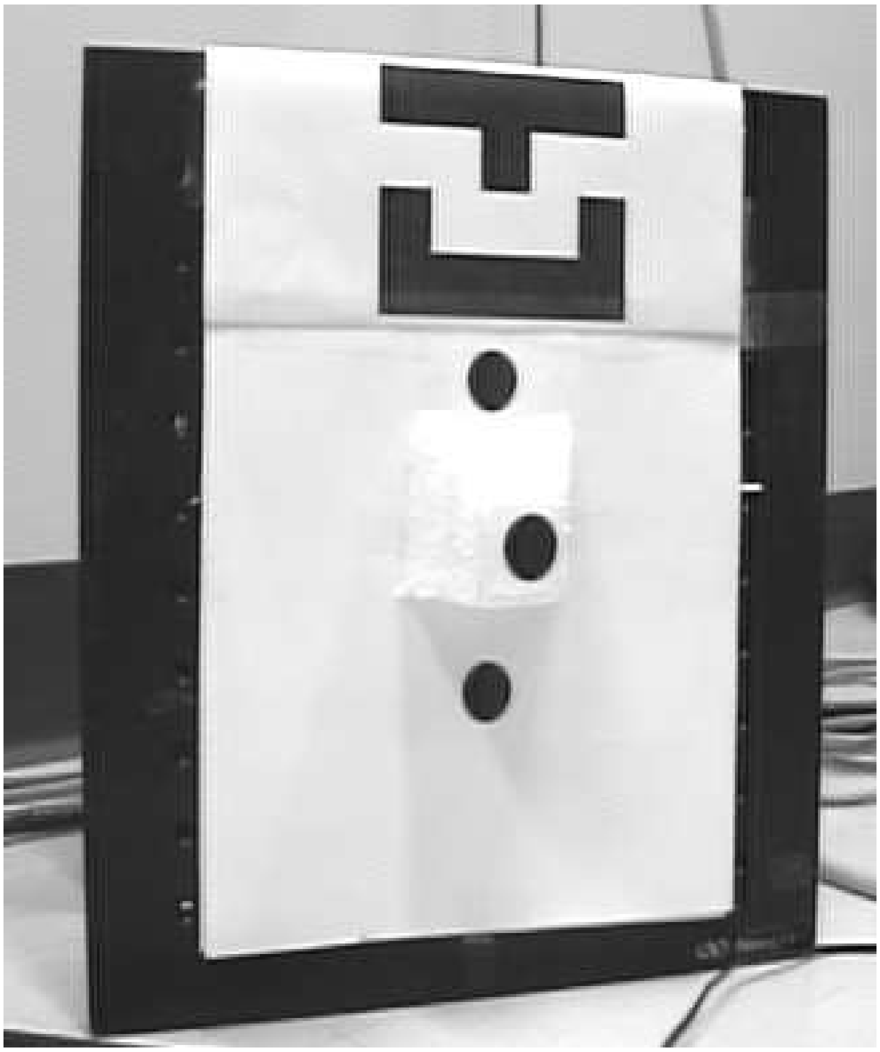

Figure 6: Landmark

dle and high-level vision processes, independent of the processing cycle of the interpretation process or the data production cycle of the early vision process.

References

[Alb92] James S. Albus. RCS: A reference model architecture for intelligent control. *IEEE Journal on Computer Architectures for Intelligent Machines*, pages 56–59, May 1992.

[BKL+93] R. Barman, S. Kingdon, J.J. Little, A.K. Mackworth, D.K. Pai, M. Sahota, H. Wilkinson, and Y. Zhang. DYNAMO: real-time experiments with multiple mobile robots. In *Intelligent Vehicles Symposium*, Tokyo, July 1993.

[Bre94] J. Brewster. Real-time onboard localization with landmarks. Master's thesis, The University of British Columbia, Vancouver, BC, 1994.

[Bro87] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Transactions on Robotics and Automation*, 2:14–23, 1987.

[Con92] Jonathan H. Connell. SSS:a hybrid architecture applied to robot navigation. In *Proc. IEEE Conf. on Robotics and Automation, 1992*, pages 2719–2724, May 1992.

[de93] G.A. den Boer and etal. An exception handling model applied to autonomous mobile robots. In *Proc. Intelligent Autonomous Systems 3*, pages 297–306, 1993.

[FB89] Anita Flynn and Rodney Brooks. Building robots: Expectations and experiences. In *Proceedings of the IEEE Intl. Rob. and Sys. Conf.*, Tsukuba, Japan, September 1989.

[Fir92] R. James Firby. Building symbolic primitives with continuous control routines. In *1st Int. Conf. on AI Planning Systems*, pages 61–69, 1992.

[LBKL91] J. J. Little, R. A. Barman, S. J. Kingdon, and J. Lu. Computational architectures for responsive vision: the vision engine. In *Proceedings of CAMP-91, Computer Architectures for Machine Perception*, pages 233–240, December 1991.

[LK93] James J. Little and Johnny Kam. A smart buffer for tracking using motion data. In *Proc. Workshop on Computer Architectures for Machine Perception*, pages 257–266, December 1993.

[MG93] Larry Matthies and Pierrick Grandjean. Stochastic performance modeling and evaluationf of obstacle detectability with imaging range sensors. Technical Report 93–11, NASA-JPL, March 1993.

[Sah94] Michael K. Sahota. Reactive deliberation: An architecture for real-time intelligent control in dynamic environments. In *Proc. 12th National Conference on Artificial Intelligence*, pages 1303–1308, 1994.

[SMV+93] P.M. Sharkey, D.W. Murray, S. Vandevelde, I.D. Reid, and P.F. McLauchlan. A modular head/eye platform for real-time reactive vision. *Mechatronics Journal*, March 1993.

[SST86] Steven A. Shafer, Anthony Stentz, and Charles E. Thorpe. An architecture for sensor fusion in a mobile robot. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 2002–2011, April 1986.

[ZM92] Y. Zhang and A. K. Mackworth. Modeling behavioral dynamics in discrete robotic systems with logical concurrent objects. In S. G. Tzafestas and J. C. Gentina, editors, *Robotics and Flexible Manufacturing Systems*, pages 187–196. Elsevier Science Publishers B.V., 1992.