Autonomous Robots 6, 21–38 (1999)

AV-Shell, an Environment for Autonomous Robotic Applications Using Active Vision

JEFFREY A. FAYMAN AND EHUD RIVLIN
Department of Computer Science, Israel Institute of Technology—Technion, Haifa, Israel
jefff@virtue3d.com
ehvdr@cs.technion.ac.il

HENRIK I. CHRISTENSEN
Centre for Autonomous Systems, Numerical Analysis and Computing Science, Royal Institute of Technology, S-100 44 Stockholm Sweeden
hic@nada.kth.se

Abstract. In this paper, we present a system called the Active Vision Shell (AV-shell) which provides a programming framework for expressing and implementing autonomous robotic tasks using perception and action where perception is provided by active vision. The AV-shell is a system with a powerful interactive C-shell style interface providing many important capabilities including: (1) architectural support; (2) an abstract interface enabling interaction with a wide variety of devices; (3) a rich set of visual routines; and (4) a process composition framework. The utility of the AV-shell is demonstrated in several examples showing the relevance of the AV-shell to meaningful applications in autonomous robotics.

Keywords: active vision, autonomous robots, agent architecture, visual routines

1. Introduction

Vision is the most versatile sensory system possessed by humans. Eyes provide the ability to perceive and act in unconstrained dynamic environments. It is clear that machine systems must possess similar visual capabilities in order to behave autonomously in these environments. Active vision is an approach to machine vision which attempts to provide these capabilities. Active vision, which was first introduced in (Bajcsy, 1985) and later explored in (Aloimonos et al., 1987) and (Ballard, 1991), is defined as the explicit control of the perception system to improve robustness and eliminate ill-posed conditions. It has been shown that by actively controlling the visual system, many classic computer vision problems become easier to solve (Aloimonos et al., 1987). Enabling robotic systems with active vision capabilities therefore enhances their ability to act autonomously in dynamic environments.

Due to the complex nature of visual perception, an autonomous system possessing active visual capabilities requires flexible tools and powerful architectures to integrate and control perception and action at multiple levels. Swain and Stricker (1993) call such systems "Agent Architectures", which they define as follows: "a programming framework for expressing perception and action routines, and the algorithms, data structures and techniques/methodology for combining them in a robot agent to achieve a task".

In this paper, we present an agent architecture called the Active Vision Shell (AV-shell) which can be viewed as a programming framework for expressing perception and action routines in the context of autonomous robotics using active vision. The AV-shell is a system with a powerful interactive C-shell style interface and features including architectural support, an abstract interface enabling interaction with a wide variety of devices, a rich set of visual routines, and a process

Figure 1. The Technion Robotic Head is an example of an active vision device whose control requires a tight coupling between perception and action. Information provided by a variety of visual cues is used to modify system parameters such as fixation point according to task requirements.

composition framework. Using the AV-shell, users can easily specify tasks requiring autonomous behavior. The example of such a task used throughout the paper is a manufacturing process where an active vision supervisor provides information to a variety of manipulators with hand mounted cameras to autonomously construct widgets whose parts are delivered on a conveyor belt in an unconstrained manner (i.e., location and timing of part delivery is not specified and must be determined autonomously by the system in real-time). The active vision supervisor provides part trajectory information to guide the manipulators and the manipulators use information extracted from their hand mounted cameras to grasp the parts.

The active vision supervisor (Technion Robotic Head (TRH)) is illustrated in Fig. 1. An active vision device such as the TRH is normally engaged in one of four behaviors: (1) *Saccades*, which are used for rapid change of fixation point from one position to another; (2) *Fixation*, which is responsible for centering the image of an object; (3) *Pursuit* which is employed for tracking objects during motion and; (4) *Stabilization* which is used for tracking structures that occupy a dominating part of the field of view. These behaviors are in turn driven by a variety of visual cues which will be discussed later.

An agent architecture must possess several key features to enable an autonomous robotic system with active visual capabilities to successfully achieve the types of tasks described above:

Architectural support. The complex nature of integrating perception and action requires architectural support. The architectural design should provide both real-time and interactive support, should be reactive, flexible, and should be hardware independent to promote portability and code reuse.

Device integration and hardware independence. Information provided by visual cues is used to drive actions. Two types of actions can be delineated: actions serving the task such as manipulator motion towards a part, and actions aiding the visual process such as head motion during pursuit. The agent architecture must provide a suitable platform capable of motor control and integration of a variety of device types including manipulators, heads and sensing devices. Additionally, for portability, it is desirable to provide an interface that is applicable to a wide variety of device types such as (Allen et al., 1991; Christensen et al., 1994; Gosselin and Hamel, 1994; Pahlavan and Eklundh, 1993).

Hierarchy of visual based cues for perception-action tasks. Perception-action tasks based on visual perception rely heavily on image based cues. In the case of active vision, these cues can be both monocular (e.g., accommodation) and binocular (e.g., disparity). Routines for extracting these cues are used as building blocks of more complex visual routines, thus an effective agent architecture must provide an appropriate hierarchically organized set of routines for carrying out a variety of vision based tasks. Additionally, as many methods exist for implementing these cues, and the fact that active vision is a developing area, the agent architecture should provide the ability to easily integrate new methods.

Module composition. As the routines mentioned above are organized hierarchically, tools for composing them into behaviors and effectively scheduling the behaviors in response to dynamic events in the environment (such as part arrival in the example outlined above) are required. These tools should enable the formal analysis of action plans.

Easy access to the capabilities of supported devices. The capabilities discussed above should be accessible to users in a convenient and clear manner. As such, an agent architecture should provide an interface to its capabilities along with the flexibility of writing programs which use them. The ability to perform simulations without endangering physical devices is also desirable.

The AV-shell is a system designed to provide these features. The remainder of this paper is organized as follows: In Section 2 we expand on the requirements of a system capable of supporting autonomous robotic

applications using active vision and briefly touch on how the AV-shell addresses these issues. In Section 3, we review relevant related work. In Section 4 we discuss the AV-shell and its structure, and show in more detail how it meets the requirements specified in Section 2. Examples of AV-shell usage are given in Section 5 and we conclude with Section 6.

2. Requirements of a Platform Supporting Autonomous Behavior Using Vision

In this section, we expand on the requirements, touched on in Section 1, of a system which supports autonomous behaviors based on vision. The system we are presenting, the AV-shell, was developed as an attempt to meet these requirements. The AV-shell can be viewed as playing dual roles: (1) as suggesting an agent architecture framework; and (2) a platform providing the necessary tools for integrating visual perception into autonomous robot tasks.

2.1. Architectural Support

An autonomous system with visual capabilities must make continuous use of visual perception at several levels. This poses some interesting architectural problems. On one hand, perception and action must be viewed at a micro-level looking only at the vision device as it can operate independently in an perception-action cycle as evidenced by smooth pursuit or fixation. On the other hand, at the macro-level, the perceptions provided by the visual system are used to drive actions of other devices in the integrated system. Providing the tight coupling between perception and action requires the architecture to possess real-time capabilities. Additionally, the dynamic nature of real-world tasks demands reactive behavior and it is desirable for these capabilities to be provided in a flexible manner.

A typical control strategy for robotic devices consists of three levels: *task*, *command* and *joint*. At the task-level, reasoning is incorporated into the control system with methods such as knowledge-based systems, fuzzy logic and artificial neural networks. At the command level, output from the task-level is converted into a form that is suitable to the joint controllers. This usually includes both direct and inverse kinematics and Cartesian velocity mappings. The command level should provide an interactive interface so that researchers can bypass the task-level planners. Joint level controllers deal with sensor integration and servo joint control. This is the level where connection to physical devices is made. These levels are illustrated in Fig. 2.

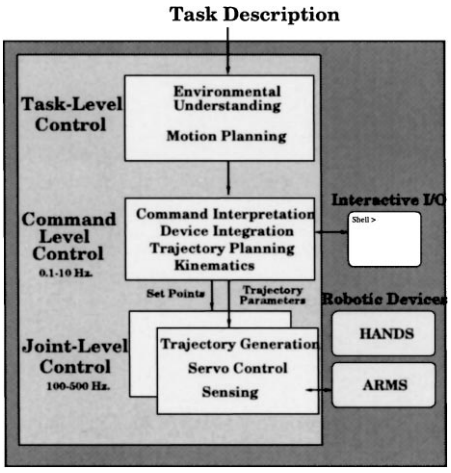


Figure 2. A typical control hierarchy for robotic manipulators includes three levels with the lowest level being the closest to physical hardware and the highest level the most removed. The command level converts data at the task level into a form usable at the joint level.

While this hierarchy is appropriate for devices such as manipulators, it is insufficient for integrating active visual perception. In principle all of the visual cues could be used independently, and the architecture suggested would perform fast task switching between the different processes. However, for many practical systems, it is advantageous to use several cues concurrently. Examples of such integrated architectures have been presented by Andersen (1996) and Uhlin (1996). In these architectures, processes are organized in a hierarchy according to their spatio-temporal scope, i.e., static image based features are used at the lowest levels of the system while more global features are used at the upper most levels of the system. An example of such a system is shown in Fig. 3.

Mechanisms for the composition of lower level visual routines into higher level behaviors are required. The higher level behaviors in turn must be composed according to changing task demands and effectively invoked. This requires, among other things, compensation of system delays.

2.2. Abstract Interface to Hardware: Achieving Flexibility and Portability

The primary function of an active vision device is to allow for active movement of the visual sensors

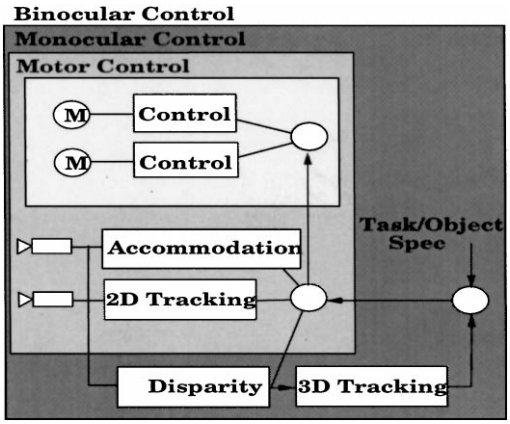


Figure 3. An example of a hierarchical architecture for active vision where processes are organized according to their spatio-temporal scope.

thereby making previously ill-posed problems well defined. This implies that the device is simply a manipulator used for moving the visual sensors. Various paradigms have been used to do this including *moveable-eyes* (Bederson et al., 1992; Gosselin and Hamel, 1994), *eye-in-hand* (Allen et al., 1991; Papanikolopoulos et al., 1991), *robotic heads* (Christensen, 1993; Pahlavan and Eklundh, 1993), *head-in-hand* (Pretlove and Parker, 1993) and *camera or head mounted on a mobile platform* (Christensen et al., 1994; Crowley and Christensen, 1995). Figure 4

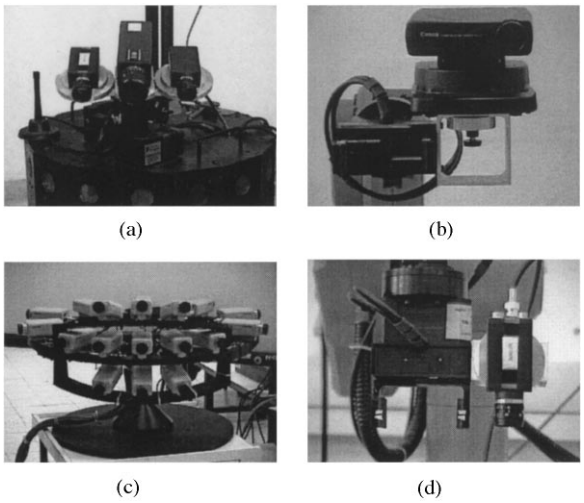


Figure 4. An abstract interface to hardware enables the same software to operate the wide range of active vision devices in existence. Examples of the variety of devices include; (a) trinocular stereo head mounted on mobile platform; (b) pan-tilt unit mounted on robotic arm; (c) 20 camera panoramic head; (d) eye-in-hand.

illustrates several of these configurations active in the Intelligent Systems Laboratory at the Technion.

It is desirable to provide an abstraction for a general interface to this wide range of devices. At the same time, we would like encapsulation to enable transparent use of existing control software. This naturally leads to an object oriented approach and because we conceptually have difficulties supporting both perception and action in one device, we believe that it is beneficial to view the vision sensors and the mechanisms used to move them separately. We define an "Active-vision device" (AVD) to include two parts: the vision sensor, which we refer to as "camera" and the device used for moving the camera, which we refer to as "head". Cameras and heads can be thought of as generic sensors and actuators, respectively. By doing this, several benefits are realized. Firstly, the mechanism will be applicable to stationary vision systems as well; these are simply cameras without the ability to move. Secondly, by decoupling cameras and heads, the "vision" problems and the "active" problems of active vision can be studied individually or together. Finally, this approach leads to portability as we will show in Section 5.

2.3. Basic Set of Visual Routines

The main objective of the visual system is to insure that the object of interest (or changing objects of interest) is continuously projected onto the area of the eye with the highest resolution (fovea centralis), and that the projected images are of high quality.

In (Ullman, 1984), Ullman discusses the idea that different image processes should share elemental operations implying that new combinations of basic operations can be assembled to meet new computational goals. Swain and Stricker (1993) advocate this idea when they say that an important area of active vision research is in developing a vocabulary of visual routines to be used as the building blocks for vision-based tasks in situated robotics. These views lead to a layered, hierarchical structure for visual routines where higher level routines are made up of lower level routines.

The visual routines provided by the AV-shell are organized hierarchically into three "levels". The first (high) level defines the visual capabilities of the system such as those described for active vision systems in Section 1. At the second (medium) level, basic process routines are defined which support the high-level routines. These include *monocular cues* which are optics

or image derived cues obtained from a single camera, *dynamic monocular cues* the most commonly used of which is image motion, *binocular cues* which ensure coordination between the two cameras and *dynamic binocular cues* which make it possible to perform tracking in 3D (Ayache, 1991). At the third (low) level, the AV-shell provides a set of functions which are typically used in image processing.

2.4. Composition Framework

While the routines outlined previously constitute a rich set of visual routines, it is important to provide tools for composing them into continuously running perception-action processes as well as integrating them with the robotic components of the system. This requires among other things, the ability to specify process concurrency. For example, consider one of the subtasks in the manufacturing process outlined in Section 1 and detailed in Section 5. A stereo robotic head locates arriving parts on the conveyor belt and provides part trajectory information to the arm. The arm is guided to grasp the part using trajectory information from the head and visual information from the arm mounted camera. The subtask includes the following events: (1) *peripheral motion detection* (a cue to the system that a part is arriving on the conveyor belt); (2) *fixation*; (3) *smooth pursuit*; (4) *trajectory estimation*; (5) *part tracking by arm camera*; and (6) *part grasping*. This process is continuously repeated by the system as new parts arrive. Figure 5 illustrates, in the form of a flowchart, these events and their temporal relationships (the line leaving the part tracking box is dashed to represent the condition of successful part tracking before part grasping).

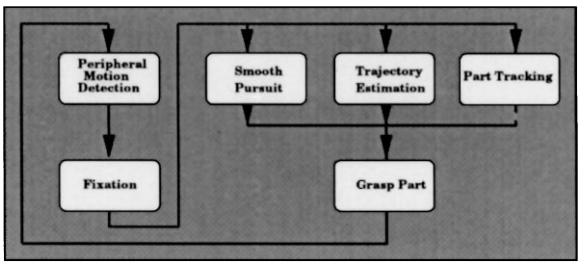


Figure 5. Flowchart showing the temporal sequence of events required in the subtask of part location, tracking and grasping. The example illustrates the need for a mechanism of specifying concurrency, conditional execution and iterative execution.

Peripheral motion detection is followed sequentially by fixation. After the part is fixated by the head, smooth pursuit, trajectory estimation and part tracking by the arm are executed in parallel. Then, sequentially, after the arm is successfully tracking the part, the arm grasps the part. The process composition mechanism provided by the system should be rich enough to capture these relationships in a straightforward manner.

3. Related Work

The AV-shell as an integrated environment provides many features necessary for implementing meaningful applications in autonomous robotics. While various other systems have appeared in the literature, the AV-shell is one of the first to provide a platform based on active vision.

3.1. Generic Architectures

A large number of efforts have addressed issues in robotic architectural design, examples include (Albus et al., 1987; Arkin and Balch, 1997; Brooks, 1996; Firby, 1989). However, most of these efforts are directed towards a specific system or application. Only the NASREM architecture (Albus et al., 1987) was designed to be truly general purpose, however it is a general framework that has not been directly implemented. Characteristically, many of these architectures have adopted a particular paradigm for system organization, i.e., NASREM is strictly hierarchical, while many of the more recent architectures such as AuRa (Arkin and Balch, 1997) and RAP (Firby, 1989) have adopted a behavior based (and often highly reactive) paradigm. For the implementation of a tool to be used in experimentation, it is not immediately obvious that the system should be tailored to a specific paradigm. Preferably the tool should be able to accommodate both paradigms to enable flexibility and to put as few constraints as possible on the user.

3.2. Architectures for Vision

Recently, Andersen (1996) and Pahlavan et al. (1996) have shown specific architectures for real-time active vision, in which a layered model is adopted. The low level modules are responsible for motor control and proprioceptive feedback, while vision based processes are organized according to their spatio-temporal scope (i.e., image based cues are subordinate to scene (3D)

related cues). This organization was chosen to enable real-time operation using a set of redundant/non-robust cues, i.e., if the 3D cue(s) is unavailable the system will default to using image based cues. These systems have a very high performance, but they were tailored for the application at hand, and as such they are difficult to port/change to new applications.

Crowley and Christensen (1995) discuss an architecture and system called VAP-SAVA for the integration and control of a real-time active vision system. The SAVA system, developed as part of the VAP project, was designed to be a "general purpose" test bed for real-time active vision systems. The system is built as a hierarchy of processes that communicate by pairwise high bandwidth communication channels and a common blackboard type of medium. The system architecture is strictly hierarchical, so processes at the same level of abstraction must be integrated into a single module. This imposes strict rules on the programs. In contrast the AV-shell uses a more flexible framework for composition of systems, which allow for architectural flexibility, but it also implies that it is more difficult to provide optimized communication structures.

Firby et al. (1995) present an "Animate Agent Architecture", a "general purpose vision system" capable of supporting a wide variety of tasks. In the system, visual routines, composed of combinations of primitive visual operators, are paired with action routines. These pairs, called "reactive skills" are designed to run in tight perception-action cycles to meet real-time constraints. Reactive skills are invoked and terminated by a reactive plan executor called "RAP" which has access to the agents knowledge of the task and environment and selects the appropriate set of control and perceptual routines. Communication between RAP and the reactive skills is done through messages called "signals". It is not immediately clear what level of programming support is provided by this system, the difficulty involved in composing visual operators into visual routines or what visual capabilities are provided.

3.3. Generic Sensors

Crowley et al. (1993) discussed a head control mechanism called the "virtual head" which provides a protocol general enough to map onto the kinematic structure of many physical heads. In their controller, all device specific parameters are encoded in a low-level module called the translator. This work is similar in nature to ours in that both are trying to provide a general mechanism for controlling a variety of heads, however, Crowley et al. are concerned only with stereo robot heads whereas we are interested in more general methods applicable to a wide variety of AVDs. This can be seen as a first step in creating an AV-shell like interface.

The idea of a vocabulary of primitive processes is discussed in (Ullman, 1984) where Ullman presents a set of five operations that he believes are important for extracting shape properties and spatial relations among objects and object parts. He discusses the importance of a fixed set of powerful basic operations together with the means for combining them into different routines. We similarly define a set of routines, however their definition is from an active-vision/purposive point of view. We also provide the tools necessary for combining these routines into higher level processes.

3.4. Behavior Based Systems

Brooks (1996) presents the "Subsumption Architecture". The Subsumption Architecture proposes a "vertical" rather than "horizontal" decomposition of a mobile robot control system. Rather than the traditional approach of decomposing the control problem into pieces, solving the subproblems for each piece and then composing the solutions, the Subsumption Architecture decomposes the problem according to desired behaviors or "levels of competence". The levels of competence are organized as concentric circles of competences where the lowest level (inner) competence fully implements the lowest level of behavior provided by the system such as avoiding contact with objects for a mobile robot. Higher levels can both view and inject data into their immediately lower levels effectively inhibiting their outputs. The basic computational unit in the Subsumption Architecture is the "module" which is modeled by a finite state machine with a number of input and output lines which can be connected. The AV-shell differs from the Subsumption Architecture in that the AV-shell is meant to provide a tool and programming environment in which users can experiment with a variety of architectural decompositions rather than proposing a new decomposition. Additionally, the AV-shell approaches the intelligent system problem from an Active Vision perspective. Therefore, many of the particular characteristics of a system based on active vision are provided.

MacKenzie et al. (1997) have designed an interactive system for specification of robot missions. The

system has been used successfully for both in- and outdoor navigation. The system uses a visual programming paradigm for the specification of missions. Basic building blocks are simple behaviors that can communicate using signals. The framework allows a hierarchy of behaviors through use of "collections" of behaviors, it can thus accommodate both hierarchical and reactive/behavior based architectures. It is not immediately obvious how simple it is to introduce new devices into the system, and the system does not appear to support real-time response, as is needed for actuator control.

Clark and Ferrier (1988) propose a control system for shifts of focus of attention on a binocular camera head based on the MDL (Motion Description Language) of Brockett (1998). According to the MDL, complex motions are specified as "modes" (i.e., triples specifying the adaptive nature of the control). Modes are specified at a higher level providing device independence. Clark and Ferrier used the MDL to shift the focus of attention of their head based on the saliency of features. Our work differs in that we provide a higher level standardized interface and language for controlling active vision devices. When viewed in this context, the "modes" of Clark and Ferrier can be used to provide an interface between our language and the low-level controllers, however, we use other mechanisms which will be presented later in this paper.

Process composition by representing task/plans as networks of processes was first proposed by Lyons and Arbib (1985) where the Robot Schema model is discussed. Kosecka et al. (1995) adopt Robot Schemas and show how one can synthesize a finite state machine supervisor which serves as a discrete event controller. Elementary behaviors appropriate in the domain of an "intelligent delivery agent" are described and experiments in robot navigation are presented. Our work also makes use of the Robot Schemas model, however, our elementary behaviors include behaviors from the active vision domain.

4. Meeting the Requirements—The AV-Shell

We designed the AV-shell with the goal of meeting the requirements previously outlined. The active vision components of the AV-shell provide a standard interface to a wide variety of AVDs. Integrating active vision into autonomous systems is accomplished by combining the AV-shell with a powerful robotic system called the Robot Shell (R-shell) (Fayman, 1990; Vuskovic et al., 1988). The R-shell provides tools necessary for controlling and coordinating various robotic devices such as manipulators and dextrous robotic hands. However, it does not possess active vision capabilities. The AV-shell was designed to fill this gap. Together, the AV-shell and the R-shell provide a powerful tool for implementing complex autonomous perception-action tasks. In this section, we discuss the structure of the AV-shell in more detail. We begin with a brief overview of the R-shell. then show how the AV-shell addresses the requirements presented in Section 2.

4.1. Robot Shell

The R-shell is an interactive program written in C under the UNIX operating system. It interprets and executes robotics related commands called "R-shell commands" in much the same way that a shell such, as c-shell, interprets and executes system related commands. Robotic data in the R-shell environment are entered, manipulated and displayed through the use of these commands. Included in the set of R-shell commands are an extensive set of algebraic operations which are used extensively in robotics work.

R-shell commands can be entered interactively or placed in R-shell scripts and executed by simply typing the script file name followed by its actual parameters. Thus scripts provide procedural abstraction in R-shell. Scripts can call other scripts and can call themselves providing nesting and recursion. The R-shell script language provides a full set of flow control statements.

4.2. The AV-Shell

4.2.1. Augmented Architecture for AVDs. In the case of AVDs, the AV-shell uses an architectural structure similar to that of Fig. 2, but augmented with two additional levels. The new levels are called the "Active Vision Integration Level" (AVIL) and the "Active Image Processing Level" (AIPL). The augmented hierarchy is shown in Fig. 6.

According to the diagram, the joint-level is now responsible for providing pre-attentive image processing and lens control in addition to servo control. The AIPL performs filtering and fusion to provide data to the set of attentive routines found in the AVIL. The AVIL in turn is responsible for composing the higher level primitives according to changing requirements and compensating system delays. All three of these levels are accessible

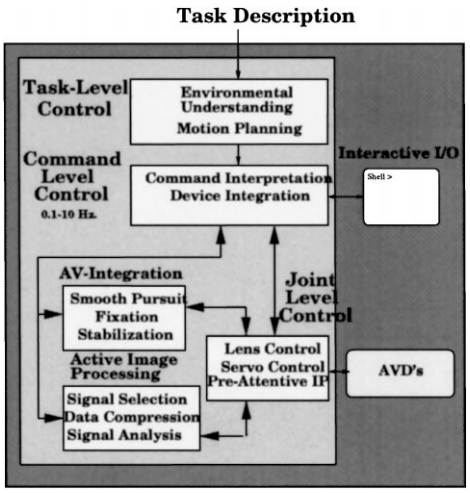


Figure 6. The augmented hierarchy with provisions for handling requirements specific to AVDs. Included are the active vision integation level and the active image processing level.

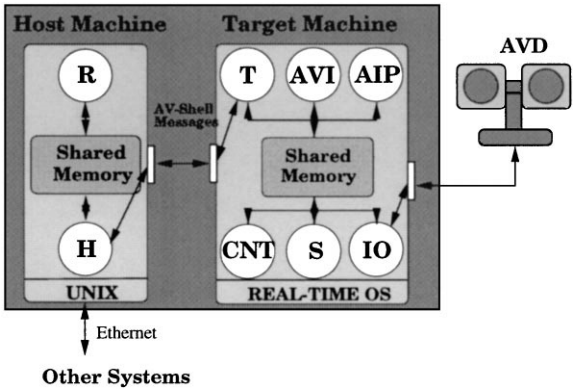


Figure 7. The logical system architecture consists of a host machine and a target machine which are decoupled due to their differing timing requirements.

from the command interface at the command-level facilitating development and simulation.

4.2.2. Portable Logical Architecture. By the nature of the different speed requirements necessary for interactive and real-time processing at the command and AVIL, AVPL and servo levels, these components should be decoupled. The interactive and developmental components of the AV-shell should make use of well known development tools. The C programming language and Unix operating system provides just such tools, therefore, these components of the system are implemented in C under Unix.[1] Unix, however, is not a real-time operating system and is not applicable for the time requirements of the AVIL, AVPL and low-level subsystems. Therefore, a separate AVIL, AVPL and low-level module is employed which uses hardware/software applicable to real-time activities. This module is called the "Target Machine". Figure 7 shows the logical system configuration.

In this configuration, the Host machine runs two processes, the **R** process executing R-shell/AV-shell which interacts with the user and generates messages whenever a command relating to a camera or a head is issued, and the **H** host server process which communicates with the target machine and updates shared host machine local variables.

The target machine runs the following processes: **T** which communicates with the host and uses the messages to update shared target machine local variables which are then interpreted by other target processes, the **AVI** active vision integration process which is responsible for composing the primitive pre-attentive active vision routines into data usable by the higher level attentive routines and performing data reduction, the **AIP** active image processing process which is responsible for compensating system delays and determining the appropriate composition of higher level attentive AV processes given changing requirements, the **CNT** control processes which provide servo control, the **S** sensor processes which monitor selected data and periodically update the host machine, **AIP** and **AVI** processes about changes, and finally, the **IO** process which is used for communicating with physical devices being controlled.

The host/target configuration discussed above for AVDs applies to other device types such as manipulators and mobile platforms. In these cases, processes on the target machine reflect the requirements of the particular device type being controlled. Figure 8 illustrates a system in which the AV-shell is configured to control an AVD, a manipulator and a mobile platform.

4.2.3. Standard Representation for AVDs. The AV-shell uses an object oriented approach to provide abstraction for a generalized interface to a wide range of devices. A three level hierarchy consisting of *class*:*type*:*instance* is used. The root of the hierarchy is a device class. The shell defines classes for manipulators, robotic heads, cameras and dextrous robotic hands. Classes contain information which, for example, determine the kinematic layout of a manipulator (i.e., Denevit-Hartenberg parameters), and for a head, the class specifies characteristics possessed by the head

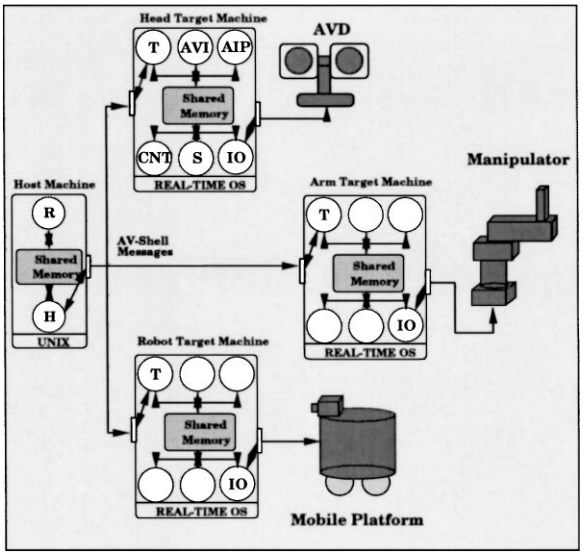


Figure 8. An integrated system. The host machine acts as an interface and coordinator for three different mechanical systems, in this example, an active vision device, a mobile platform and a manipulator.

such as independent pan, independent tilt and cyclotorsion.

Data specific to a given class define its type, and an instance contains data specific to a particular device. The hierarchy provides the ability to mix and match manipulators, cameras and heads to flexibly construct many AVDs. For example, a stereo robotic head can be defined and attached to a manipulator to create a head-in-hand. All values specified are converted to device coordinates at the lowest level. More examples of the variety of device types supported by the shell are given in conjunction with the examples in Section 5.

4.2.4. Basic Visual Routines. At the highest level of the visual routine hierarchy are the activity routines. The AV-shell provides routines supporting activities such as *fixation*, *pursuit*, *stabilization*, and *saccade* discussed in Section 1.

At the medium-level, the AV-shell includes a set of basic process routines which are used to implement high-level activities. The implementation details of these routines is not specified by the AV-shell. Therefore, the same interface is applicable without concern for underlying implementation and algorithms can be easily interchanged. This feature is important as active vision is a developing area in which many paradigms enjoy a distinct popularity in the research community and researchers would like the benefit of being able to easily experiment with a variety of algorithms. For example, many techniques exist for tracking such as Active Deformable Models (Terzopoulos and Szeliski, 1993), Optical Flow (Barron et al., 1994) and Template Matching (Ballard and Brown, 1982). Each of these techniques can be implemented and easily integrated into the system. We will demonstrate this capability in Section 5.

The basic process routines include *dynamic accommodation*, which provides sharp high quality images by focus adjustments and is used as a depth cue, *aperture control* with which we are able to adjust for changes in lighting giving high contrast images over a range of lighting conditions, *vergence/disparity estimation* with which we can keep the fixation distance consistent with the target of fixation, *peripheral motion detection* with which we are able to detect motion in some unknown area of the scene, and *foveal motion detection* with which we can detect motion in a small region centered on the image plane.

At the low-level, the AV-shell provides a set of functions which are typically used in image processing such as *laplacian*, *convolution*, *epipole*, *edge detection* and *histogramming*. For a more detailed description of all AV-shell commands (see Fayman et al., 1995).

4.2.5. Composition Framework. The AV-shell adopts a model proposed in (Lyons, 1993; Lyons and Arbib, 1985) called Robot Schemas (*RS*) for specifying process concurrency and the temporal and structural dependencies inherent in complex perception-action tasks.

The *RS* model was selected as it possessed two main characteristics that we deemed necessary: (a) the ability to order the processes sequentially, in parallel, preemptively and recurrently; and (b) the ability to communicate between processes. While other models could have been used such as *communicating sequental processes* (Hoare, 1985) or *discrete event systems* (Kosecka and Bogoni, 1994), the representation would have been less compact and the mapping to a development system with interactive support would be more complicated. *RS* provides a convenient port automata model which satisfied our requirments.

Table 1 summarizes the *RS* composition operators.[2] In the *RS* model, communication channels between concurrent processes are called “ports”. Messages are written to, and read from ports. A port-to-port *connection relation* can be specified as an optional third parameter in concurrent composition. This connection

Table 1. Summary of the *RS* composition operators. The operators provide an elegant process algebra capable of expressing the complexities of autonomous behavior based on perception and action.

1. *Sequential Composition*: $T = P; Q$. The composition T behaves like the process P until that terminates, and then behaves like the process Q (regardless of P's termination status).
2. *Concurrent Composition*: $T = (P \mid Q)^c$. The composition T behaves like P and Q running in parallel and with the input ports of one connected to the output ports of the other as indicated by the port-to-port connection map c. This can also be written as $t = (|_{i \in I}\ P_i)^c$ for a set of processes indexed by I.
3. *Conditional Composition*: $T = P\langle v \rangle : Q_v$. The composition T behaves like the process P until that terminates. If P aborts, then T aborts. If P terminates normally, then the value v calculated by P is used to initialize the process Q, and T then behaves like Q_v.
4. *Disabling Composition*: $T = P\#Q$. The composition T behaves like the concurrent composition of P and Q until either terminates, then the other is aborted and T terminates. At most one process can stop; the remainder are aborted.
5. *Synchronous Recurrent Composition*: $T = P\langle v \rangle :: Q_v$. This is recursively defined as
$$P :: Q = P : (Q; P :: Q).$$
6. *Asynchronous Recurrent Composition*: $T = P\langle v \rangle :: Q_v$. This is recursively defined as
$$P :: Q = P : (Q \mid (P :: Q)).$$

relation specifies a set of couples $op \mapsto ip$ indicating that port *ip* and *op* are connected.

Composition of primitive routines into complex activities is achieved by traversing a parse tree built from an expression in *RS* notation. For instance, the activity of *pursuit* is represented by the parse tree shown in Fig. 9.

The application of pursuit on an AVD such as a stereo robotic head is implemented as a continuous perception-action process composed of vergence control, foveal motion detection, and dynamic accommodation. These tasks execute in parallel and their results are composed into a single move command on the condition that they all have executed successfully. The pursuit example above illustrates the *RS* operators of *synchronous recurrent composition* (:;) and *disabling composition* (#).

Using *RS* notation, we can readily see how the high-level activities of fixation and, pursuit can be easily specified (stabilization is similar to pursuit): Fixation is initialized with a saccade to the fixation point

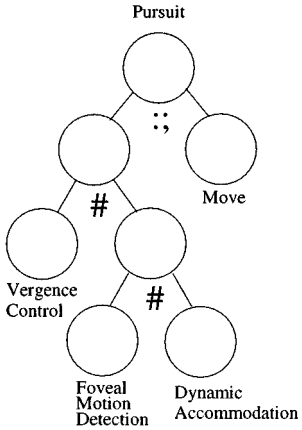


Figure 9. The AV-shell parse tree representing the activity of smooth pursuit. The tree contains nodes composed with the *RS* operators of *synchronous recurrent composition* (:;) and *disabling composition* (#).

followed by continuous vergence control driven by disparity and accommodation cues.

$$\text{fixation} = \text{saccade}; ((\text{disparity} \mid \text{accommodation}) :: (\text{vergence control}))$$

In pursuit, vergence, foveal motion detection and dynamic accommodation are used to continuously drive motion of the vision sensor.

$$\text{pursuit} = (\text{vergence} \,\#\, \text{foveal motion} \,\#\, \text{accommodation}) :; \text{move}.$$

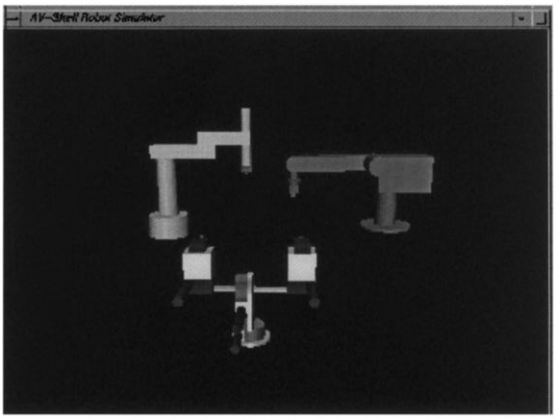
Finally, using *RS*, we can elegantly specify the subtask discussed in Section 2.4:

$$\text{capture} = (\text{peripheral} ; \text{fixate}) :: ((\text{pursuit} \mid \text{trajectory} \mid \text{track part}) : \text{grasp}).$$

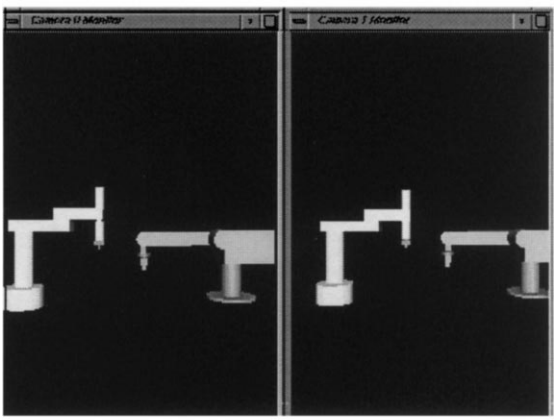
4.3. The AV-Shell Integrated Programmers Environment

The user of the AV-shell has at her/his disposal a rich language for working with a wide range of devices including standard robotic manipulators, dextrous robotic hands, active vision devices, cameras and arbitrary objects. For a thorough description of all commands, see (Fayman et al., 1995). The logical architecture presented in Section 4.2.2 insures that the system can simultaneously manage multiple devices and can be ported to a variety of physical architectures as we will show in Section 5, enabling shell software to be portable.

In addition to a text based interface, the AV-shell provides a 3D graphical subsystem. Using the subsystem, users can control simulated versions of any devices supported by the shell. The graphic simulator includes a simulated camera that enables users to "look" on simulated scenes from the viewpoints of multiple simulated cameras. Cameras can be attached to devices in the environment such as heads or manipulators and output from the rendering pipeline can be fed to algorithms implementing any visual process enabling control closed by vision. Figure 10(a) and (b) shows an example scene from the AV-shell graphics simulator in which two manipulators and a head (AdeptOne, Scorbot ER-9 and TRH head) are being controlled. The TRH has two attached cameras viewing the scene. The views from these cameras are rendered in the two "monitor" windows of Fig. 10(b).



(a)



(b)

Figure 10. The users environment includes a graphic simulation subsystem where both robotic and visual algorithms can be tested. (a) illustrates a view of a simulated environment containing a stereo robotic head and two manipulators. (b) shows the view of environment as seen from the two simulated cameras mounted on the head.

5. Experiments

In this section, we present our experiments. Two experiments illustrate the shell's usefulness in implementing meaningful applications in autonomous robotics as well as its portability.

5.1. Using the AV-Shell: An Example Application

5.1.1. Experimental Scenario. The scenario used in the first experiment consists of a manufacturing plant occupied by a variety robotic devices including manipulators, a stereo robotic head, and a mobile platform. The manufacturing process requires a set of manipulators to construct a widget. Parts of the widget are delivered to the first robot (a Scorbot ER9) on a conveyor belt. After working on the part, the robot places the part on a mobile platform (a Nomad 200) which carries it to the second robot (an AdeptOne). The second robot takes the part from the mobile platform and completes the manufacturing process. No assumptions about the timing of part delivery or placement of parts on the conveyor belt (for the first robot) or mobile platform (for the second robot) are made (i.e., determination of these parameters must be made autonomously by the system). Therefore, the robots have cameras attached to their end-effectors to help guide part location and grasping. Figure 11(a) and (b) show the two manipulators in the scenario. Figure 11(b) also shows the part delivery conveyor belt (simulated by a toy train (part) on a track (conveyor belt) in the experiment).

In order to supervise the entire process and to provide useful information, a stereo robotic head (supervisor) is mounted in the plant and is used to assist various tasks in the manufacturing process (the supervisor is illustrated in Fig. 1). For example, the head computes trajectory information of the parts moving on the conveyor belt to aid the Scorbot in grasping the parts and also computes trajectory information of the mobile platform to help guide it from the workcell of the Scorbot to the workcell of the Adept. Figure 11(c) shows the Scorbot placing the completed part on the mobile platform and Fig. 11(d) shows the mobile platform delivering the part to the Adept. The physical layout of the manufacturing plant is illustrated in Fig. 11(e).

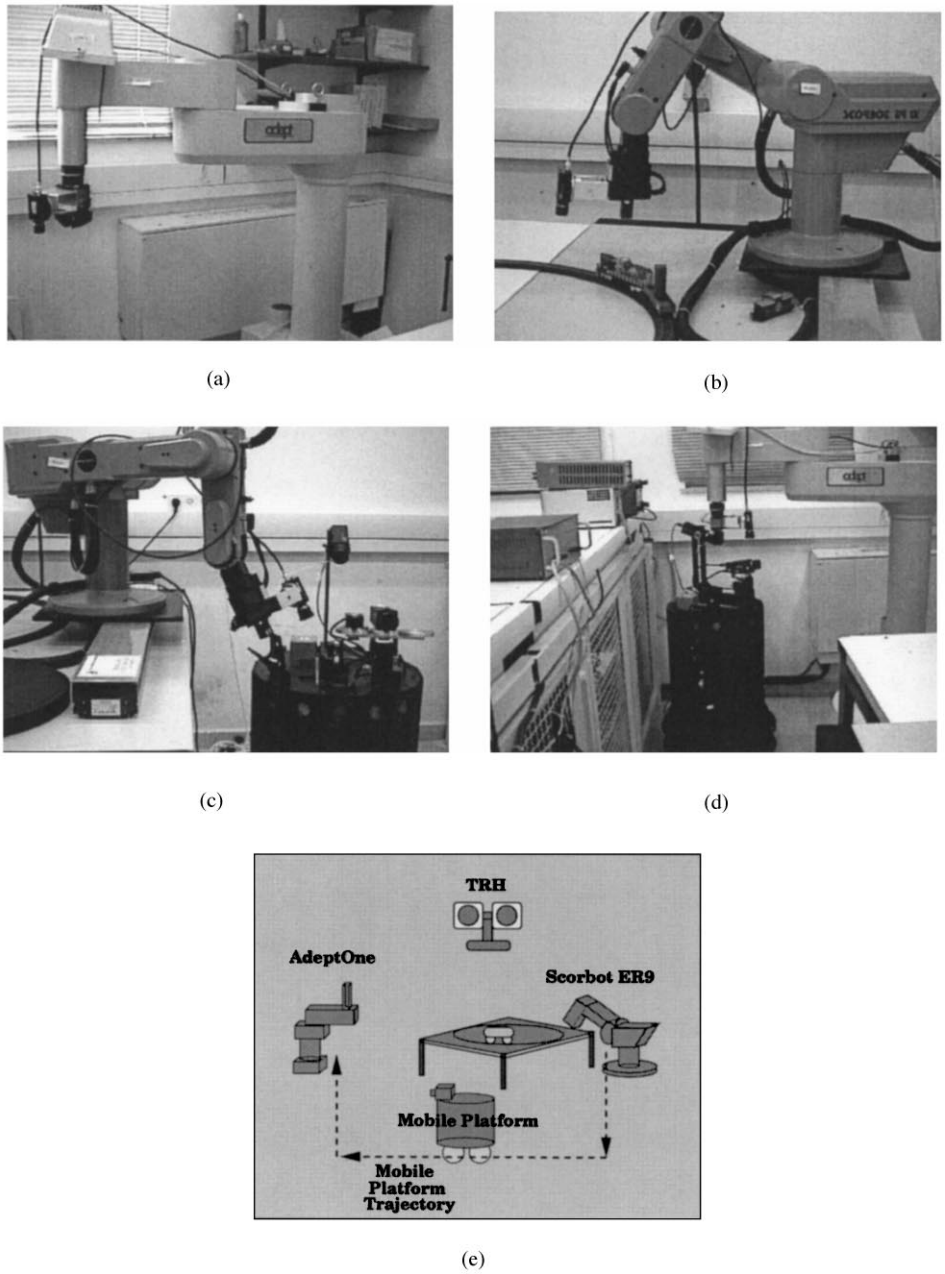


Figure 11. The scenario consists of two manipulators: an AdeptOne (a), and a Scorbot ER9 (b) which cooperate in the manufacture of a widget. Parts are delivered to the Scorbot on a conveyor belt (b) (depicted as a train on a track carrying the part in the figure). The Scorbot uses information provided by the active vision supervisor (see Fig. 1) and its hand mounted camera to pick parts from the conveyor belt. The Scorbot transfers completed parts to the Adept by placing them on the mobile platform (c) which delivers them to the Adept (d) for widget completion. The physical layout of the plant is shown in (e).

In this experiment we highlight several of the shell's key characteristics including the ability to coordinate a variety of device types, process composition, and hardware/software abstraction. To show this, we implement several subtasks of the scenario.

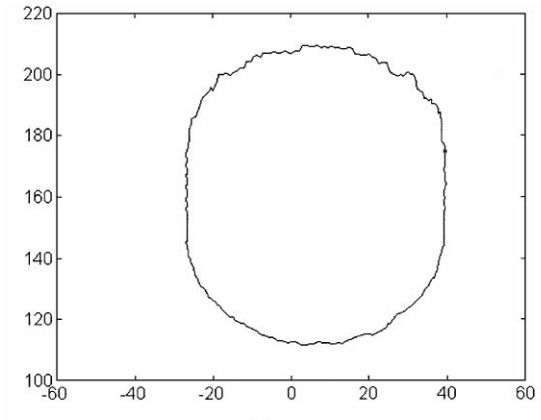
5.1.2. Head/arm Coordination. In the first subtask, the stereo head supervisor tracks the moving part computing 3D position coordinates and building a part trajectory. The head first performs peripheral motion detection looking for the part to enter the field of view.

```
fixation     = motion -p ; saccade
pursuit      = motion -f 40 :; move -3
findandtrack = fixation ; (pursuit |
               choosegrasp pickup)
pickuppart   = loc pickup ; grip -c ;
               trans -z -90
findandtrack : pickuppart
```

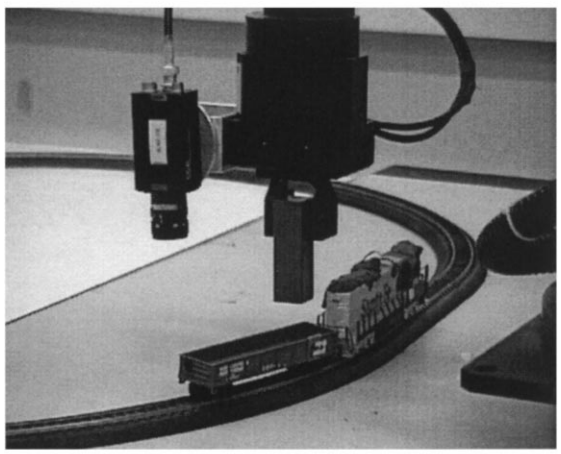
Figure 12. The head/arm coordination task as written in AV-shell notation. The task is written in two sections. The first four lines express subtasks and the last line invokes the task. The subtask, *findandtrack*, is the composition of three subtasks: *fixation*, *pursuit* and *choosegrasp*. Once the part is fixated, *pursuit* and *choosegrasp* are executed in parallel in order to track the part (*pursuit*), and build a trajectory and choose a grasping location (*choosegrasp pickup*). *Fixation* is defined as the sequential composition of peripheral motion detection (for finding the moving part in some part of the periphery (*motion -p*)) and *saccade*. *Pursuit* is defined as the synchronous recurrent composition of foveal motion detection (*motion -f 40* where 40 is the size of the fovea in pixels) and movement (*move -3* performs the tracking in 3D). The subtask *pickuppart* uses the R-shell commands *loc*, *grip* and *trans*. *Loc pickup* drives the end-effector of the manipulator to the pickup location computed by *choosegrasp*. Once location is complete, *grip -c* closes the gripper on the part and finally, *trans -z -90* causes the manipulator to lift the part off the conveyor belt to a height of 90 cm.

Once motion is detected, a saccade is executed and the part is fixated. Smooth pursuit tracking follows along with trajectory estimation. A part grasping location along the trajectory is chosen. Finally, the arm is commanded to grasp the part at the selected grasping location. The task is expressed in AV-shell notation as shown in Fig. 12.

Image space positional information of the part was independently computed for each head camera using SSD correlation based template matching. The template size was 40×40 pixels, and correlation was performed on 150×150 pixel regions of interest centered around the previous known part location in each image. The cameras were independently controlled to keep the object stabilized in the image frames using the *pursuit* AV-shell command. To this end, the kinematic model of the head was used to compute the 3D position of the part in cyclopean coordinates and these coordinates were smoothed by a Kalman filter. Coordinates were updated at a rate of approximately 15 Hz. Figure 13(a) illustrates the trajectory of the part tracked by the system as it completes one cycle, and Fig. 13(b) shows the manipulator successfully grasping the part. The data gathered is 3D, however the part moves along the track placed on the table and the motion is therefore planar, here we will present only 2D coordinates (on the x-y plane) although full 3D information is available. We ran the experiment several times, each time moving the track so that the part arrived at a different location. The system was able to compensate for the changes and succeeded in grasping the part each time.



(a)



(b)

Figure 13. The figure shows the trajectory of the part obtained from the head (a) and the manipulator grasping the part (b).

5.1.3. Mobile Platform Trajectory Guidance. In the second subtask, we use information provided by the stereo head supervisor to guide the mobile platform from the work-cell of the Scorbot to the work-cell of the Adept. As shown in Fig. 11(e), the (known) path followed by the mobile platform consists of three linear segments. Once the Scorbot has placed a part on the mobile platform, the system signals the platform to

```
pursuit      = motion -f 40 :; move -3
guide        = ((pursuit | forward) :;
               notarrived x y 25)
x = 160
y = 325
guide ; turn -90
x = -150
y = 325
guide ; turn -90
x = -150
y = 100
guide ; stop
```

Figure 14. The mobile platform trajectory guidance task as written in AV-shell notation. The task is written in two sections. The first two lines express subtasks and the next nine lines invoke the task. The subtask *pursuit*, is defined in the same way as in Fig. 12. The subtask *guide* is defined as the concurrent composition of pursuit and *forward* (the mobile platform moves along the trajectory and is pursued by the head) both in synchronous recurrent composition with *notarrived* which responds positively as long as the current position of the robot is not within a 25 cm radius of x, y. *Turn* causes the mobile platform to rotate the specified number of degrees.

begin moving in a straight line along the first segment. The supervisor tracks the progress of the mobile platform and when head data indicates that the platform has arrived at the intersection of the first and second trajectory segments (which is known in advance), the mobile platform is instructed to turn -90° and continue moving along the second segment. The same is repeated for the intersection of the second and third trajectory segments. When the platform arrives at a predefined location in the work-cell of the Adept, it is commanded to stop.[3] The details of the tracking algorithm are similar to those described for the previous experiment. The task is expressed in AV-shell notation as shown in Fig. 14, and Fig. 15 shows the trajectory followed by the mobile platform.

5.1.4. Visual Servoing Example. In the third subtask, we use visual information provided by the camera mounted on the Adept's end-effector to visually servo the Adept. The goal is to insure that the part arriving on the mobile platform is located under the Adept's end-effector so that it is graspable.

The purpose of this experiment is to demonstrate the transparent interface to underlying software provided by the AV-shell. In the previous examples, we used template matching based tracking to track the moving part and mobile platform with the head. Here, we use a tracking algorithm based on active contour models

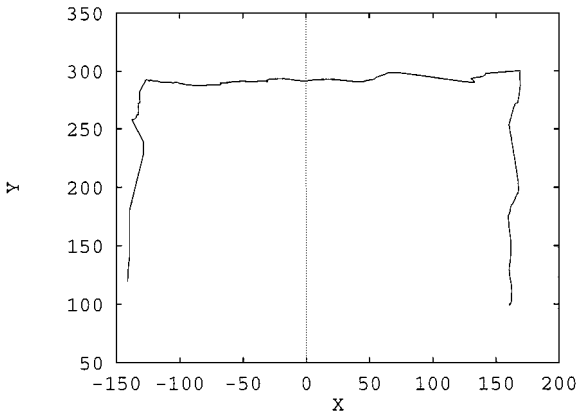


Figure 15. Trajectory followed by the mobile platform.

(snakes) to visually servo the Adept. In a similar manner, any other implemented algorithm can be used. For instance, in the Intelligent Systems Laboratory at the Technion, we have experimented with five different tracking algorithms (details on these algorithms can be found in (Fayman et al., 1996)). Any of these algorithms could have been used in this experiment instead of snakes.

We placed the part on the mobile platform and drove the platform to four locations. The locations were chosen to be within the reachable workspace of the Adept, visible by the mounted camera, and consistent with possible arrival positions of the mobile platform to the Adept's workspace. For snake implementation, we used a library of routines provided by the GSNAKE API which are based on the Generalized Active Contour Model (Lai, 1994). A simple calibration technique was used to relate the coordinates of the part's location in pixel coordinates (the center of gravity of the computed snake) to the Cartesian coordinates of the manipulator. The AV-shell script used in the example is shown in Fig. 16. The part used in the experiment is shown in Fig. 17(a). The part as seen from the hand mounted camera is shown in Fig. 17(b), and the contour as computed by the system is shown in Fig. 17(c). Tracking results are shown in Fig. 17(d).

```
snaketrack :; loc uv
```

Figure 16. Visual servoing expressed in AV-shell notation. *Snaketrack* invokes the low-level snake tracker which returns, in AV-shell vector variable *uv*, the coordinates of the center of gravity of the localized and minimized snake. The *loc* command causes end-effector motion to the manipulator coordinate system equivalent of screen coordinate *uv*.

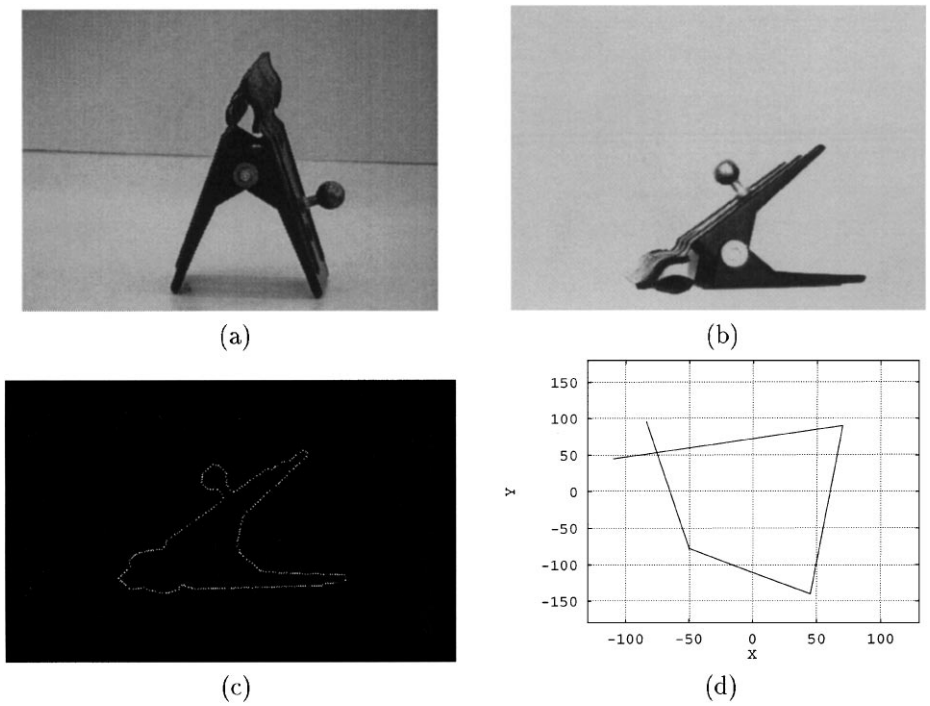


Figure 17. The part used in the experiment (a). The part as seen from the Adept's arm mounted camera (b) and the Generalized Active Contour of the part computed by the system (c). The trajectory followed by the Adept (d).

5.2. Portability

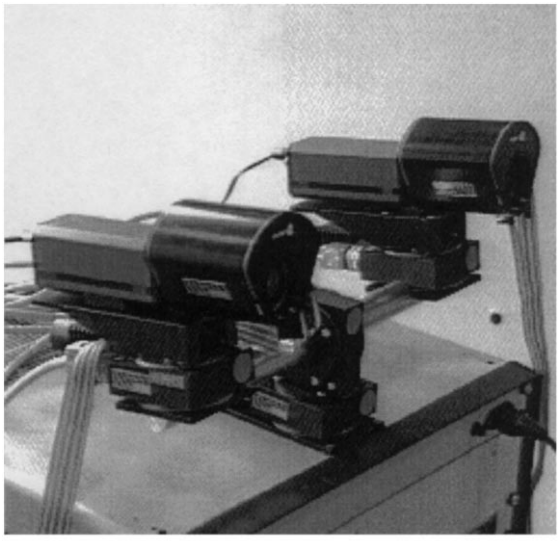
In the last experiment, we illustrate the shell's portability and transparent interface to underlying hardware by executing the same tasks on two different architectures supporting two different robotic heads. We ported the system to two different heads, the AUC camera head at Aalborg University in Denmark and the TRH at Technion—Israel Institute of Technology. The architectures employed at these sites for controlling the heads are inherently different, however, the logical architecture of Section 4.2.1 mapped well onto both physical architectures and using the AV-shell, we were able to execute the same task at both sites with the same AV-shell script.

Figure 18(a) shows the AUC camera head and Fig. 18(b) shows the TRH camera head. The architecture of Aalborg University consists of a VME rack housing five boards used to control the AUC camera head while the architecture a the Technion consists of a 90 MHz Pentium PC running DOS which is connected via ethernet to a Sun workstation running Unix and the AV-shell. Figure 18(c) illustrates the Aalborg architecture and Fig. 18(d) illustrates the Technion architecture. For a thorough description of the hardware of these architectures (see Fayman et al., 1995).

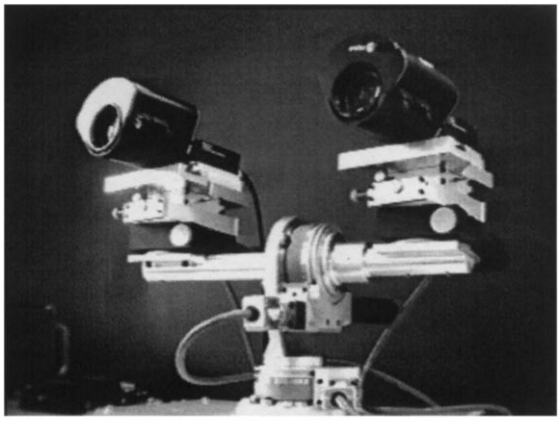
Using the AV-shell, we implemented on both platforms the scenario subtask of locating moving parts and tracking them. This task is similar to that described in (Bradshaw et al., 1994) and discussed in Section 2.4. The task consists of peripheral motion detection, fixation and smooth pursuit. The idea is to wait for a moving part to enter the field of view of the camera, fixate on the part and proceed to track it. Figure 19 shows the task as it is expressed in AV-shell notation and Fig. 20 shows the resulting parse tree. The experiment verified that the shell can be used to implement identical tasks with hardware and architecture transparency.

6. Conclusion

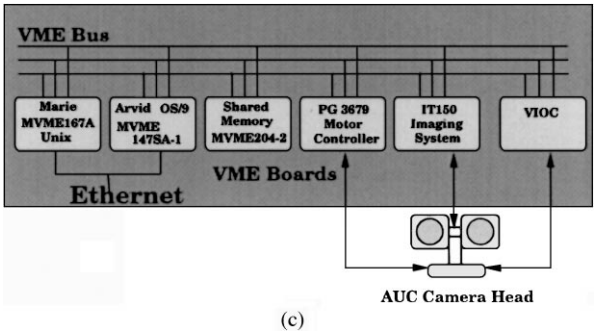
In this paper, we presented the AV-shell, an agent architecture which provides a programming framework for expressing perception and action routines in the context of autonomous robotics using active vision. We discussed issues raised when integrating the perceptive capabilities of active vision and the active capabilities of other robotic devices to achieve goals autonomously.



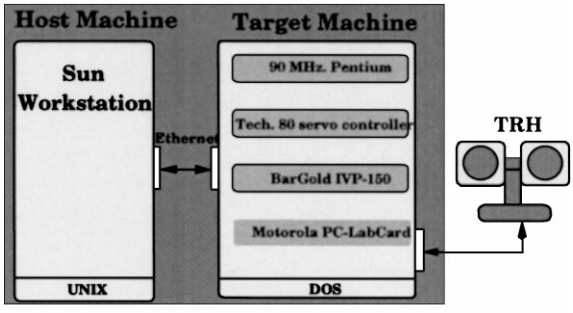
(a)



(b)



(c)



(d)

Figure 18. In this figure, we illustrate the two heads (a) Aalborg head, and (b) Technion head as well as the two architectures upon which they run (c) Aalborg architecture and (d) Technion architecture. The AV-shell was ported to both platforms and we were able to execute the same task on both platforms using the same AV-shell script.

```
fixation     = motion -p ; saccade
pursuit      = motion -f 40 :; move -3
findandtrack = fixation ; pursuit
```

Figure 19. Expression of task in AV-shell notation. The findtrack task consists of the sequential composition of two subtasks: fixation and pursuit. Fixation in turn is the sequential composition of peripheral motion detection (*motion -p*) for finding the moving part in some area of the field of view and saccade. Once the part is fixated, pursuit is invoked. Pursuit is the synchronous recurrent composition of foveal motion detection (*motion -f 40* where 40 is the size of the fovea in pixels) and movement (*move -3* performs the tracking in 3D).

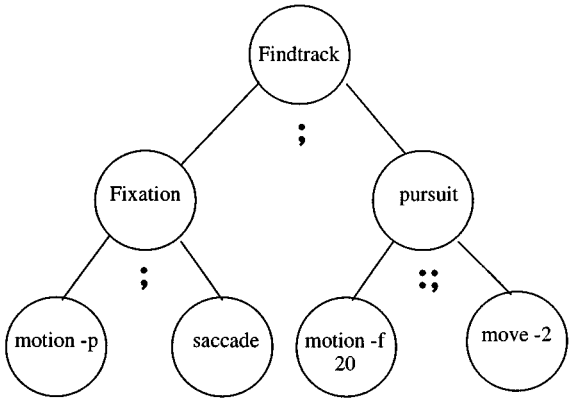


Figure 20. Parse tree resulting from the *findandtrack* AV-shell task.

These issues include architectural support, an abstract interface to hardware providing flexibility and portability, a well-defined set of visual routines and the ability to compose them along with action routines into continuously running perception-action tasks. We have shown how the AV-shell addresses these issues in such a way as to provide an integrated programming environment for achieving autonomous behavior.

We presented a logical architectural design which decouples the interactive and real-time components of the system and provides porability, and a uniform interface to multiple device types. The AV-shell abstracts underlying hardware from the user level providing a transparent interface to a wide range of device types. Additionally, we discussed a three level hierarchically organized set of visual routines whereby lower level routines are composed into higher level processes using the Robot Schemas process algebra. The utility of the AV-shell is demonstrated in several experiments showing relevance of the shell to meaningful applications in autonomous robotics.

Future work on the AV-shell includes adding fault-tolerant capabilities to the system, fusion of redundant multi-valued behaviors to improve system reliability, and further research into the integration of modules for robust system behavior.

Acknowledgment

This work was supported in part by the European Community—Israel collaboration ECIS–003 project.

Notes

1. The use of C and Unix provide portability and are accepted in the vision and robotics communities as standard.
2. Notation in Table 1 is consistent with those of (Lyons, 1993), however, in our implementation, the operators were changed to avoid conflicts with existing operators. The modified operators are described in (Fayman et al., 1995).
3. Future work includes closed loop control of the mobile platform during trajectory following. We plan to update platform telemetry data with head data to insure that the platform moves accurately along the trajectory and guarantee that the platform arrives at the trajectory segment intersection points.

References

Albus, J.S., McCain, H.G., and Lumia, R. 1987. Nasa/nbs standard reference model for telerobot control system architecture (nasrem). Technical report, NBS Technical Note 1235 Robot Systems Division, National Bureau of Standards, Robot Systems Division.

Allen, P.K., Yoshimi, B., and Timcenko, A. 1991. Real-time visual servoing. In *Proc. of the IEEE International Conference on Robotics and Automation*, Sacramento, California, pp. 851–856.

Aloimonos, Y., Weiss, I., and Bandyopadhyay, A. 1987. Active vision. *International Journal on Computer Vision*, pp. 333–356.

Andersen, C.S. 1996. A framework for control of a camera head. Ph.D. Thesis, Laboratory of Image Analysis, Aalborg University, Denmark.

Andersen, C.S. 1996. A framework for control of a camera head. Technical report, Aalborg University.

Arkin, R.C. and Balch, T. 1997. Aura: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):175–189.

Ayache, N. 1991. *Artificial Vision for Mobile Robots: Stereo Vision and Multisensory perception*, MIT Press.

Bajcsy, R. 1985. Active perception vs. passive perception. In *Proc. of the Third IEEE Workshop on Computer Vision*, Bellaire, Michigan, pp. 55–59.

Ballard, D.H. 1991. Animate vision. *Elsevier Artificial Intelligence*, 48:57–86.

Ballard, D.H. and Brown, C.M. 1982. *Computer Vision*, Prentice-Hall: Englewood Cliffs, NJ.

Barron, J.L., Fleet, D.J., and Beauchemin, S.S. 1994. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77.

Bederson, B.B., Wallace, R.S., and Schwartz, E.L. 1992. Two miniature pan-tilt devices. In *Proc. of the IEEE International Conference on Robotics and Automation*, Nice, France, pp. 658–663.

Bradshaw, K.J., McLauchlan, P.F., Reid, I.D., and Murray, D.W. 1994. Saccade and pursuit on an active head/eye platform. *Image and Vision Computing*, 12(3):155–163.

Brockett, R.W. 1998. On the computer control of movement. In *Proc. of the IEEE International Conference on Robotics and Automation*, Philadelphia, PA.

Brooks, R.A. 1996. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23.

Christensen, H.I. 1993. A low-cost robot camera head. *International Journal of Pattern Recognition and Artifical Intelligence*, 7(1):69–87.

Christensen, H.I., Kirkeby, N.O., Kristensen, S., Knudsen, L., and Granum, E. 1994. Model-driven vision for in-door navigation. *Robotics and Autonomous Systems*, pp. 199–207.

Clark, J.J. and Ferrier, N.J. 1988. Modal control of an attentive vision system. In *Second International Conference on Computer Vision*, Tampa, Florida, pp. 514–523.

Crowley, J.L., Bobet, P., and Mesrabi, M. 1993. Layered control of a binocular camera head. *International Journal of Pattern Recognition and Artifical Intelligence*, 7(1):109–122.

Crowley, J.L. and Christensen, H.I. 1995. *Vision as Process*, Springer-Verlag.

Fayman, J., Rivlin, E., and Christensen, H. 1995. The active vision shell. Technical Report CIS 9510, Technion—Israel Institute of Technology.

Fayman, J.A. 1990. Medium-level control of robot hands. Master's thesis, San Diego State University, Department of Mathematical Sciences, San Diego, CA.

Fayman, J.A., Pirjanian, P., Christensen, H.I., and Rivlin, E. 1996. Exploiting redundancy of purposive modules in the context of

active vision. CIS Report 9616, Technion—Israel Institute of Technology.

Firby, R.J. 1989. Adaptive execution in complex dynamic domains. Technical Report YALEU/CSD/RR#672, Yale University.

Firby, R.J., Kahn, R.E., Prokopowicz, P.N., and Swain, M.J. 1995. An architecture for vision and action. In *Proc. of the International Joint Conference on Artificial Intelligence*, Montreal, Canada, pp. 72–79.

Gosselin, C.M. and Hamel, J-F. 1994. The agile eye: A high-prformance three-degree-of-freedom camera-orienting device. In *Proc. of the IEEE International Conference on Robotics and Automation*, San Diego, CA, pp. 781–786.

Hoare, C.A.R. 1985. *Communicating Sequential Processes*, Prentice Hall.

Kosecka, J., Bajcsy, R., and Christensen, H.I. 1995. Discrete event modelling of visually guided behaviors. *International Journal of Computer Vision. Special Issue on Qualitative Vision*, 12(3):295–316.

Kosecka, J. and Bogoni, L. 1994. Application of discrete event systems for modeling and controlling robotic agents. In *Proc. of the 1994 International Conference on Robotics and Automation*, San Diego, CA.

Lai, K.F. 1994. Deformable contours: Modeling, extraction, detection and classification. Ph.D. Thesis, Electrical Engineering, University of Wisconsin-Madison.

Lyons, D.M. 1993. Representing and analyzing action plans as networks of concurrent processes. *IEEE Transactions on Robotics and Automation*, 9(7):241–256.

Lyons, D.M. and Arbib, M.A. 1985. A task-level model of distributed computation for sensory-based control of complex robot systems. In *IFAC Symposium, Robotic Control*, Barcelona, Spain.

MacKenzie, D.C., Arkin, R.C., and Cameron, J.M. 1997. Multiagent mission specification and execution. *Autonomous Robots*, 4(1):29–52.

Pahlavan, K. and Eklundh, J.O. 1993. Heads, eyes and head-eye systems. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(1):33–49.

Pahlavan, K., Uhlin, T., and Eklundh, J.O. 1996. Dynamic fixation and active perception. *International Journal of Computer Vision*, 17:113–135.

Papanikolopoulos, N., Khosla, P.K., and Kanade, T. 1991. Vision and control techniques for robotic visual tracking. In *Proc. of the IEEE International Conference on Robotics and Automation*, Sacramento, California, pp. 857–864.

Pretlove, J.R.G. and Parker, G.A. 1993. The surray attentive robot vision system. *International Journal of Pattern Recognition and Artifical Intelligence*, 7(1):89–107.

Swain, M.J. and Stricker, M.A. 1993. Promising directions in active vision. *International Journal of Computer Vision*, 11(2):109–126.

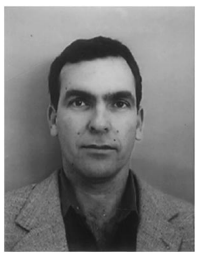
Terzopoulos, D. and Szeliski, R. 1993. *Computer Vision*, chapter Tracking with Kalman Snakes, The MIT Press, pp. 3–20.

Uhlin, T. 1996. Fixation and seeing systems. Ph.D. Thesis, Kungliga Tekniska Högskolan, S-100 44 Stockholm.

Ullman, S. 1984. Visual routines. *Cognition*, 18:97–159.

Vuskovic, M.I., Riedel, A.L., and Do, C.Q. 1988. The robot shell. *International Journal of Robotics and Automation*, 3(3):165–175.

Jeffrey A. Fayman received the M.Sc. degree in computer science from San Diego State University in 1990. He received the D.Sc. degree in computer science from the Technion—Israel Institute of Technology in 1998. Dr. Fayman is currently employed at Virtue 3D, Inc.



Ehud Rivlin received the B.Sc. and M.Sc. degrees in computer science and the M.B.A. degree from the Hebrew University in Jerusalem. He received the Ph.D. from the University of Maryland. Currently, he is an Assistant Professor in the Computer Science Department at the Technion—Israel Institute of Technology. His current research interests are in machine vision and robot naviation.

Henrik I. Christensen received his M.Sc. and Ph.D. degrees in Electrical Engineering from Aalborg University in 1987 and 1989, respectively. He was during the period 1989–1992 a research associate working on control and system integration. In 1992 he was appointed Associate Professor at Aalborg University. During 1996 he was a visiting professor at the GRASP Laboratory, University of Pennsylvania. He is presently Scientific Director of the Centre for Autonomous Systems, Royal Institute of Technology, Stockholm, Sweden. Dr. Christensen has published more than 90 papers on vision and robotics. He has also edited three books on active vision, experimental environments and integrated vision systems.