Dynamic Memory: Architecture for Real Time Integration of Visual Perception, Camera Action, and Network Communication

T. Matsuyama S. Hiura T. Wada K. Murase A. Yoshioka
Dept. of Intelligent Science and Technology
Graduate School of Informatics, Kyoto University
Sakyo, Kyoto, JAPAN 606-8501
e-mail: tm@i.kyoto-u.ac.jp

Abstract

In a Cooperative Distributed Vision *system, a group of communicating* Active Vision Agents *(AVA, in short, i.e. real time image processor with an active video camera and high speed network interface) cooperate to fulfill a meaningful task such as moving object tracking and dynamic scene visualization. A key issue to design and implement an AVA rests in the dynamic integration of Visual Perception, Camera Action, and Network Communication. This paper proposes a novel dynamic system architecture named* Dynamic Memory Architecture*, where perception, action, and communication modules share what we call the* Dynamic Memory*. It maintains not only temporal histories of state variables such as pan-tilt angles of the camera and the target object location but also their predicted values in the future. Perception, action, and communication modules are implemented as parallel processes which dynamically read from and write into the memory according to their own individual dynamics. The dynamic memory supports such asynchronous dynamic interactions (i.e. data exchanges between the modules) without wasting time for synchronization. This no-wait asynchronous module interaction capability greatly facilitates the implementation of real time reactive systems such as moving object tracking. Moreover, the dynamic memory supports the* virtual synchronization *between multiple AVAs, which facilitates the cooperative object tracking by communicating AVAs. A prototype system for real time moving object tracking demonstrated the effectiveness of the proposed idea.*

1 Introduction

In our last paper[1], we proposed the concept of *Cooperative Distributed Vision* (CDV, in short), where a group of communicating *Active Vision Agents* (AVA, in short, i.e. real time image processor with an active video camera and high speed network interface) cooperate to fulfill a meaningful task such as moving object tracking and dynamic scene visualization. A key issue to design and implement an AVA rests in the dynamic integration of Visual Perception, Camera Action, and Network Communication (Figure 3).

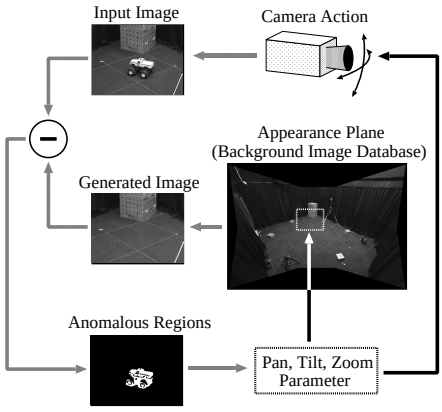


Figure 1: Active background subtraction with a fixed-viewpoint pan-tilt-zoom (FV-PTZ) camera.

This paper proposes a novel dynamic system architecture named *Dynamic Memory Architecture*, where multiple parallel processes share what we call the *Dynamic Memory*. We design and implement an AVA based on the dynamic memory architecture and demonstrate its effectiveness in real time moving object tracking.

Real time moving object tracking by an active video camera(s) is a key technology for visual surveillance and dynamics scene visualization. In [1], we developed a real time active object tracking system with a *fixed-viewpoint pan-tilt-zoom (FV-PTZ, in short) camera*: its projection center stays fixed irrespectively of any camera rotations and zoomings. The system uses an off-the-shelf active video camera SONY EVI-G20, which can be well modeled as an FV-PTZ camera.

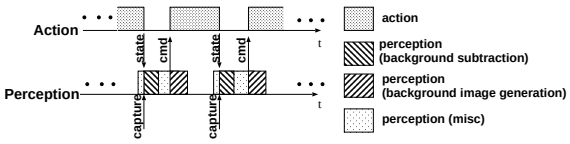


Figure 2: Dynamics of the prototype system[1]

Figure 1 illustrates the basic scheme of the active background subtraction for object tracking[1]:

1. Generate the *APpearance Plane* (APP) image of the scene; with an FV-PTZ camera, a wide panoramic image (i.e. APP image) can be easily generated by mosaicing multiple images observed by changing pan, tilt, and zoom parameters.

2. Extract a window image from the APP image according to the current pan-tilt-zoom parameters and regard it as the background image; with the FV-PTZ camera, there exists the direct mapping between the position in the APP image and pan-tilt-zoom parameters of the camera.

3. Compute difference between the background image and an observed image.

4. If anomalous regions are detected in the difference image, select one and control the camera parameters to track the selected target.

To cope with dynamically changing situations in the real world, we have to augment the basic scheme in the following two points:

1. Robust background subtraction which can work stably under varying geometric and photometric environments.

2. Flexible system dynamics to control visual perception and camera action adaptively to unpredictable object behaviors.

This paper mainly addresses the latter problem, since various robust background subtraction methods have been developed [2].

Figure 2 shows the dynamics of the system developed in [1]. While it employs a sophisticated prediction-based camera control, its fundamental dynamics is very simple; the activations of the perception and action modules are just interleaved on the temporal axis. That is, one module stays idle while the other is activated. To realize real time reactive systems, we have to make the modules run in parallel and develop a flexible dynamic interaction mechanism

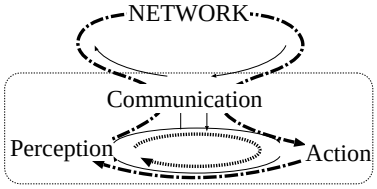


Figure 3: Module organization of an AVA and information flows among the modules.

between the modules. The dynamic memory architecture proposed in this paper supports such real time inter-module interactions.

In this paper, we first introduce the general concept and functionalities of the dynamic memory. Then, we describe an active vision system for real time moving object tracking, where perception and action modules are dynamically integrated with the dynamic memory. In the last part of the paper, we show utilities of the dynamic memory for cooperative moving object tracking by communicating AVAs, where perception, action, and communication modules are integrated with the dynamic memory and the *virtual synchronization* between multiple AVAs is realized. Quantitative dynamic characteristics of the systems are given to demonstrate their performance.

2 Dynamic Memory

In general, an intelligent system such as an AVA can be modeled as consisting of multiple modules with different functionalities and dynamics. Thus, a key issue to design and implement an intelligent system rests in the functional and dynamic integrations of the modules. Here we put our focus on the dynamic integration of the modules, since the functional decomposition of an AVA is rather straightforward: visual perception, camera action, and network communication modules.

To implement an AVA, we have to integrate three modules with different intrinsic dynamics:
Visual Perception: video rate periodic cycle
Camera Action: mechanical motions involving varying rather large delays
Network Communication: asynchronous message exchanges, where varying delays are incurred depending on communication activities over the network.

Figure 3 illustrates information flows among these modules. The problem we study here is how we can design and implement flexible dynamic information flows, i.e. dynamic interactions among the modules.

In the *Dynamic Memory Architecture*, multiple parallel processes such as perception, action, and communication modules, share what we call the *Dynamic*

Memory. The read / write operations from / to the dynamic memory are defined as follows (Figure 4):

Write Operation:

When a process computes a value v of a variable at a certain moment t, it writes (v, t) into the dynamic memory. Since such computation is done repeatedly according to the dynamics of the process, a discrete temporal sequence of values is recorded for each variable in the dynamic memory (a sequence of black dots in Figure 4). Note that since the speed of the computation varies depending on input data, the temporal interval between a pair of consecutive values becomes irregular.

Read Operation:

Temporal Interpolation: A reader process runs in parallel to the writer process and tries to read from the dynamic memory the value of the variable at a certain moment according to its own dynamics: for example, the value at T_1 in Figure 4. When no value is recorded at the specified moment, the dynamic memory interpolates it from its neighboring recorded discrete values. With this function, the reader process can read a value at any temporal moment along the continuous temporal axis.

Future Prediction: A reader process may run fast and require data which are not written yet by the writer process (for example, the value at T_3 in Figure 4). In such case, the dynamic memory predicts an expected value in the future based on those data so far recorded and returns it to the reader process. Note that as illustrated in Figure 4, multiple values may be defined by the interpolation and prediction functions, for example, at NOW and T_2 in Figure 4. We have to define the functions to avoid such multiple value generation.

With the above described functions, each process can get any data along the temporal axis freely without waiting (i.e. wasting time) for synchronization with others. That is, the dynamic memory integrates parallel processes into a unified system while decoupling their dynamics; each module can run according to its own dynamics without being disturbed by the others. This no-wait asynchronous module interaction capability greatly facilitates the implementation of real time reactive systems. As will be shown later, moreover, the dynamic memory supports the *virtual synchronization* between multiple network-connected intelligent systems (i.e. AVAs), which facilitates the dynamic cooperation among the systems.

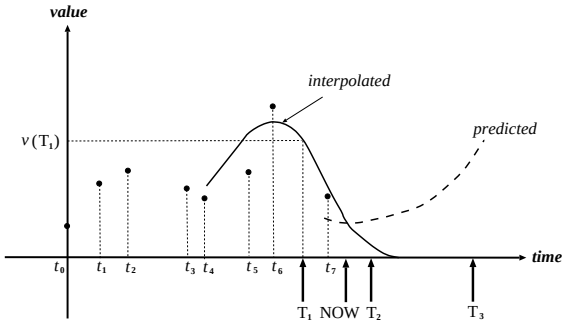


Figure 4: Representation of a time varying variable in the dynamic memory.

While the system architecture consisting of multiple parallel processes with a common shared memory looks similar to the "whiteboard architecture"[3], the critical difference rests in that the dynamic memory maintains variables whose values change dynamically along the temporal axis spanning *continuously* from the past to the *future*. Little and Kam[4] proposed an idea of the *smart buffer*, where virtual values are synthesized to dynamically coordinate parallel processes with different processing speeds. However, their idea did not include variables with dynamically changing values or their temporal interpolation and prediction. Zhang and Mackworth[5], on the other hand, proposed *constraint nets*, where variables with dynamically changing values were introduced. Their major interest, however, was in designing dynamic systems and did not refer to the dynamic integration of multiple parallel modules like the whiteboard system. Thus, the dynamic memory architecture can be regarded as an advanced dynamic system architecture integrating the whiteboard, the smart buffer, and the constraint nets.

3 Dynamic Integration of Perception and Action Modules for Real Time Object Tracking

To demonstrate the effectiveness of the above proposed idea, we developed a real time object tracking system using the dynamic memory. Figure 5 illustrates the organization of the system, where pan-tilt angles of the camera and the target object location are dynamically exchanged between the perception and action modules through the dynamic memory:

Information	Writer		Reader
Pan-Tilt Angles	Action Module	→	Perception Module
Object Location	Perception Module	→	Action Module

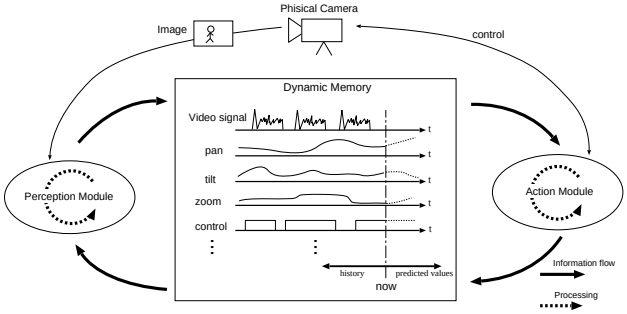


Figure 5: Real time object tracking system using the dynamic memory.

3.1 Specifications of the System

Task: The system detects and tracks the object that first comes into the scene. The gaze of the camera (i.e. the center of an observed image) is controlled to follow the centroid of the observed moving object.

Active Camera: The system employs an FV-PT camera: i.e. SONY EVI-G20 with a fixed zooming factor.

Processors: The system includes a pair of processors: PC (Pentium II 266MHz, FreeBSD) and RVS-10G (SIMD parallel video image processor with 1D array of 256 PEs). The perception and action modules are implemented by UNIX processes on PC and the dynamic memory by a shared memory between the processes. The perception process controls RVS-10G for video image capturing and processing and the action process the FV-PT camera for the gaze control.

3.2 Active Background Subtraction

Figure 6 illustrates the implementation of the real time active background subtraction. Its scheme is basically the same as the one illustrated in Figure 1.

To make this background subtraction work well, it is required to align the synthesized background image exactly with the observed image. To attain the accurate alignment, in turn, the perception module has to obtain *current* pan-tilt angles since the camera is moving continuously. This is exactly the place where the dynamic memory plays a crucial role; pan-tilt angles are measured by the action module and recorded into the dynamic memory, based on which the *current* pan-tilt angles are interpolated or predicted by the dynamic memory to answer the read request from the perception module.

Note that even with the dynamic memory, the pixel-wise exact alignment between the background and observed images is hard to attain. The perception module compensates for such small misalignments (the left column in Figure 6): several shifted versions

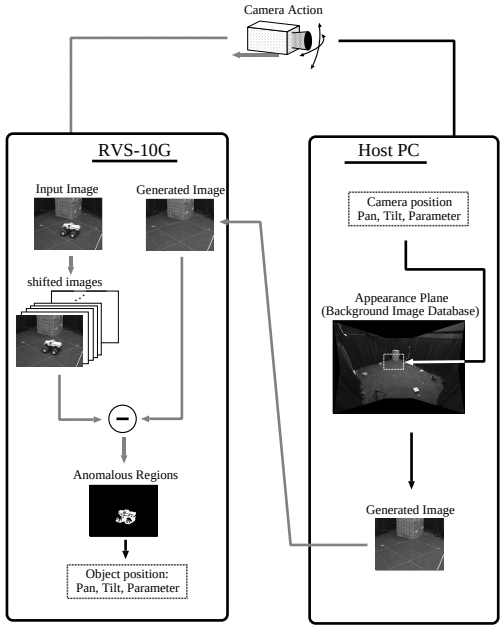


Figure 6: Real time active background subtraction.

of the observed image are generated and their differences from the background image are computed. The image with the least overall gray level difference is regarded as the result of the background subtraction.

3.3 Camera Control Method

The action module controls the FV-PT camera based on the following PID method.

1. Suppose the action module reads from the camera pan-tilt angles at t_n, $(\mathrm{P}_{cam}(t_n), \mathrm{T}_{cam}(t_n))$, where n denotes the nth control cycle.

2. Then, the module reads from the dynamic memory the location of the target object at t_n, $(\mathrm{P}_{obj}(t_n), \mathrm{T}_{obj}(t_n))$[1] In what follows, we will describe mathematical equations for the camera control and the dynamic memory management methods (Section 4) just about the pan angle, since the same descriptions apply to the tilt due to the 'symmetric' dynamics of the FV-PT camera.

3. Compute the displacement:

$$\mathrm{P}_{dis}(t_n) = \mathrm{P}_{obj}(t_n) - \mathrm{P}_{cam}(t_n) \qquad (1)$$

4. Determine the camera velocity control command $(V_P(n), V_T(n))$ by the following equations and is-

[1] With the FV-PT camera, the object location can be described in the same coordinate system as $(\mathrm{P}_{cam}(t_n), \mathrm{T}_{cam}(t_n))$.

sue it to the camera.

$$V_P(n) = K\{\frac{\mathrm{P}_{dis}(t_n)}{\Delta t} + \frac{\alpha}{(\Delta t)^2}(\mathrm{P}_{dis}(t_n) - \mathrm{P}_{dis}(t_{n-1})) + \frac{1}{\beta}\sum_{i=n-10}^{n} \mathrm{P}_{dis}(t_i)\} \quad (2)$$

where $\Delta t = t_n - t_{n-1}$ and the gains were experimentally determined: $K = 0.32, \alpha = 50, \beta = 50$.

Here again, the dynamic memory plays crucial roles in realizing stable physical camera control. Firstly, whereas the control timing t_n is determined according to the autonomous dynamics of the action module, the target object location at that specific timing can be obtained from the dynamic memory. Secondly, while the gains for the PID control were determined by purely off-line stand-alone experiments without taking into account of the dynamics of the perception module, the same stable camera control can be realized even after integrating the perception and action modules. This is because the dynamic memory guarantees that the autonomous dynamics of a module is not disturbed even if the module is integrated with other modules.

4 Implementation of the Dynamic Memory

4.1 Describing Time Varying Information

Based on the idea illustrated in Figure 4, the following descriptive method is employed in the system. Suppose a module has observed data about information INFO at t_i $(i = 1, \cdots, n)$ and the $n+1$th data is obtained at t_{n+1}. The module writes it into the dynamic memory to modify the content of the dynamic memory as follows:

$$\begin{array}{rcl} \mathrm{INFO}_i : [t_i, t_{i+1}) & & \mathrm{INFO}_j : [t_j, t_{j+1}) \\ f_i(t) & & g_j(t) \\ (i = 1, \cdots, n-1) & \Longrightarrow & (j = 1, \cdots, n) \\ \mathrm{INFO}_n : [t_n, \infty) & & \mathrm{INFO}_{n+1} : [t_{n+1}, \infty) \\ f_n(t) & & g_{n+1}(t), \end{array}$$

where $[t_i, t_{i+1})$ denotes the temporal period during which INFO_i is valid, $f_i(t)$ $(i = 1, \cdots, n-1)$ and $g_j(t)$ $(j = 1, \cdots, n)$ interpolation functions, and $f_n(t)$ and $g_{n+1}(t)$ prediction functions.

4.2 Describing the Target Object Motion

The information about the target object observed by the perception module, i.e. OBJ_i, is described in terms of its centroid position at t $(t \in [t_i, t_{i+1})$: $(\mathrm{P}_{obj_i}(t),\ \mathrm{T}_{obj_i}(t))$. Since we have no knowledge about the object motion, we model it by the constant velocity motion. Suppose observations at t_i $(i = 1, \cdots, n)$ have been done and a new data at t_{n+1}, $(\hat{\mathrm{P}}_{obj_{n+1}}, \hat{\mathrm{T}}_{obj_{n+1}})$ is written in the dynamic memory. Then, the memory rewrites the data as follows (Figure 7):

$$\begin{array}{rcl} \mathrm{OBJ}_n : [t_n, \infty) & \Longrightarrow & \mathrm{OBJ}_n : [t_n, t_{n+1}) \\ \mathrm{P}_{obj_n}(t) & & \mathrm{P}_{obj_n}(t) = \mathrm{P}_{obj_n}(t_n) + (t - t_n)\omega_p \\ & & \\ & & \mathrm{OBJ}_{n+1} : [t_{n+1}, \infty) \\ & & \mathrm{P}_{obj_{n+1}}(t) = \hat{\mathrm{P}}_{obj_{n+1}} + (t - t_{n+1})\omega_p \end{array}$$

where

$$\omega_p = \frac{\hat{\mathrm{P}}_{obj_{n+1}} - \mathrm{P}_{obj_n}(t_n)}{t_{n+1} - t_n}. \quad (3)$$

The same descriptions as above apply to the tilt.

4.3 Describing the Camera Motion

The updating process of the camera motion history in the dynamic memory involves some complications, because the action module controls the camera speed as well as measures the pan-tilt angles of the camera.

First of all we conducted experiments to model the dynamics of the FV-PT camera and found that it can be well described by the following equation in both pan and tilt controls:

$$\theta(t) = \theta_0 + t\dot{\theta}_0 \qquad (0 \le t < \tau) \quad (4)$$

$$\theta(t) = \theta_0 + t\dot{\theta}_c + (\dot{\theta}_c - \dot{\theta}_0)T\mathrm{e}^{-\frac{(t-\tau)}{T}} - (\tau + T)(\dot{\theta}_c - \dot{\theta}_0) \qquad (\tau \le t) \quad (5)$$

where θ denotes pan or tilt angle positions, and θ_0 and $\dot{\theta}_0$ means the angle position and the angular speed at $t = 0$, respectively. $\dot{\theta}_c$ denotes the camera speed control command issued at $t = 0$, which is computed by equation (2). The latency $\tau (= 70msec)$ and the time constant $T (= 50msec)$ were determined by experiments.

Suppose we have the following information about the camera motion in the dynamic memory (Figure 8A):

$$\begin{array}{l} \mathrm{CAM}_n : [t_n, \infty) \\ \qquad P_{cam_n}(t) \\ \qquad \dot{P}_{cam_n}(t), \end{array}$$

where $P_{cam_n}(t)$ and $\dot{P}_{cam_n}(t)$ denote the angle position and angular speed respectively.

When the camera speed control command $(\dot{P}_{c_{n+1}}, \dot{T}_{c_{n+1}})$ is issued at t_{n+1}, CAM is changed as follows using the camera dynamics in equations (4) and (5) (Figure 8B):

$$\begin{array}{l} \mathrm{CAM}_n : [t_n, t_{n+1}) \\ \qquad P_{cam_n}(t) \\ \qquad \dot{P}_{cam_n}(t) \end{array}$$

$$\mathrm{CAM}_{n+1} : [t_{n+1}, \infty)$$

$$P_{cam_{n+1}}(t) = \begin{cases} P_{cam_n}(t) & (t_{n+1} \le \\ P_{cam_n}(t_{n+1} + \tau) & \\ \quad +(\dot{P}_{c_{n+1}} - \dot{P}_{cam_{n+1}}(t_{n+1} + \tau))T\mathrm{e}^{-\frac{(t-t_{n+1}-\tau)}{T}} & \\ \quad +(t - t_{n+1} - \tau)\dot{P}_{c_{n+1}}(t) & \\ \quad -T(\dot{P}_{c_{n+1}} - \dot{P}_{cam_{n+1}}(t_{n+1} + \tau)) & (t_{n+1} + \tau \le t) \end{cases}$$

frame:0 frame:20 frame:40 frame:60 frame:80 frame:100 frame:120 frame:140

Figure 9: Partial sequence of the human tracking

$$\dot{P}_{cam_{n+1}}(t) = \begin{cases} \dot{P}_{cam_n}(t) & (t_{n+1} \le t < t_{n+1}+\tau) \\ (\dot{P}_{c_{n+1}} - \dot{P}_{cam_n}(t_{n+1}+\tau)) & \\ \times(1 - \mathrm{e}^{-\frac{(t-t_{n+1}-\tau)}{T}}) & \\ +\dot{P}_{cam_n}(t_{n+1}+\tau) & (t_{n+1}+\tau \le t) \end{cases}$$

After issuing the command, the action module reads from the camera the current pan-tilt angles (P', T') at $t'(> t_{n+1})$. Then, some discrepancies ($P_{err_{n+1}}$, $T_{err_{n+1}}$) may be found between the predicted and the observed pan-tilt angles (Figure 8B):

$$P_{err_{n+1}} = P' - P_{cam_{n+1}}(t'), \quad T_{err_{n+1}} = T' - T_{cam_{n+1}}(t') \tag{6}$$

To solve this conflict and generate a smooth camera motion trajectory in the dynamic memory, we modify CAM_{n+1} as follows (Figure 8C):

$$\mathrm{CAM}_{n+1} : [t_{n+1}, \infty)$$

$$P_{cam_{n+1}}(t) = P_{cam_{n+1}}(t) + P_{err_{n+1}}(1 - \mathrm{e}^{-\frac{t-t_{n+1}}{a(t'-t_{n+1})}})$$
$$\dot{P}_{cam_{n+1}} = \dot{P}_{cam_{n+1}},$$

where $a(= 1.25)$ denotes the time constant.

5 Experiments

We conducted experiments of real time human tracking in a computer room. Figure 9 illustrates a partial sequence of the tracking: upper: observed images and lower: detected object regions. Figure 10 shows the read/write access timing from/to the dynamic memory by the perception and action modules. Each vertical line denotes the read/write timing. The upper graph is about the object location data, which were written by the perception module and read by the action module. The lower one is about the pan-tilt camera position data, written by the action module and read by the perception module. We can get the following observations:

(1) Both modules work asynchronously keeping their own intrinsic dynamics.

(2) The perception module runs almost twice faster than the action module (about 66msec/cycle).

(3) Irrespectively of these mutually independent dynamics, the smooth dynamic information flows through the dynamic memory are realized without introducing any idle time for synchronization.

Figure 11 illustrates object and camera motion trajectory data written into and read from the dynamic memory, where graph1 (upper right): pan-tilt camera positions measured from the camera, graph2 (upper left): pan-tilt camera positions read from the dynamic memory, graph3 (lower left): object locations detected from observed images, and graph4 (lower right): object locations read from the dynamic memory.

Each graph includes a pair of trajectories: larger amplitude is about pan and smaller amplitude tilt. Note that object locations as well as camera positions are described in terms of (pan, tilt).

We can get the following observations:

(a) Comparing graph1 with graph2, the data density of the latter is higher than that of the former. This is because the perception module runs faster and hence reads pan-tilt camera position data more frequently. This holds also for graph3 and graph4.

(b) The smooth camera motion is realized irrespectively of the rather noisy tilt trajectory of the observed object.

(c) The camera control is well synchronized with the object motion.

(d) By overlaying graph1 (camera motion) and graph3 (object motion), we can see that the former is delayed by about 440msec from the latter. This is due to the PID control, which does not take into account delays involved in image processing and physical camera motion. To reduce the delay, we should introduce a prediction-based camera control method[6].

6 Virtual Synchronization between Multiple AVAs

In a CDV system, there exist two levels of dynamic interactions: (1) *Intra-AVA Interactions* among visual perception, camera action, and network communication modules in an AVA, (2) *Inter-AVA Interactions* among multiple communicating AVAs in a CDV system. Here we show the dynamic memory plays an important role in the inter-AVA interaction as well as the intra-AVA interaction.

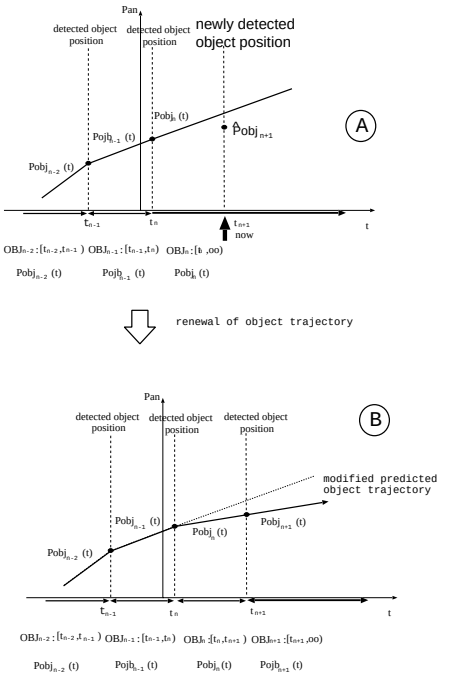


Figure 7: Updating the object motion trajectory.

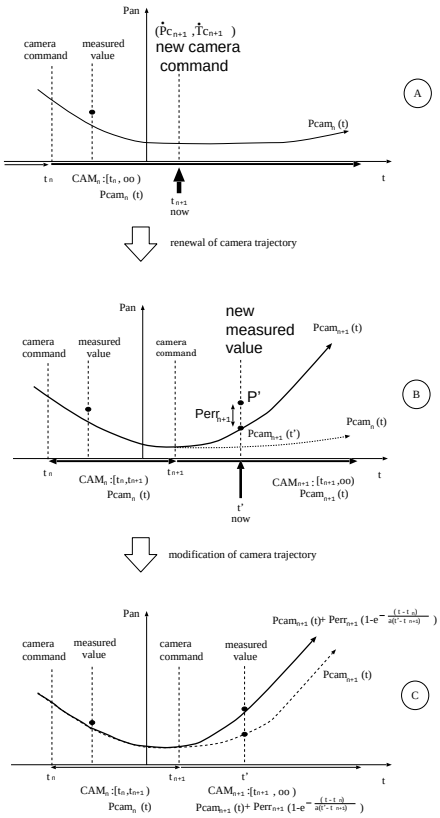


Figure 8: Updating the camera motion history.

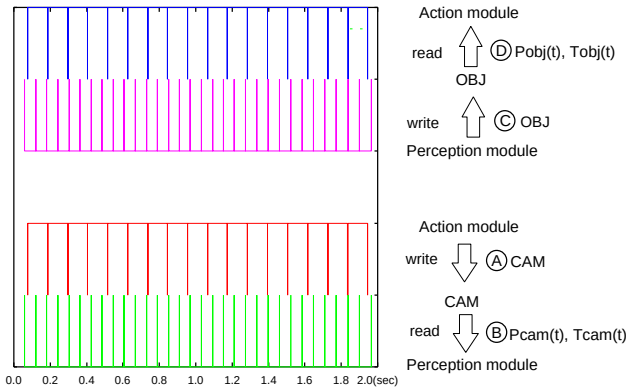


Figure 10: Access timing to the dynamic memory by the perception and action modules: Upper: object information, Lower: camera information

6.1 Synchronized Image Acquisition for Object Identification

In [1], we developed a cooperative object tracking system, where a group of communicating AVAs cooperatively track a single target object without being interfered by occluding obstacles or other moving objects (Figure 12).

To realize such cooperative gazing, the object identification should be established across multiple AVAs. In [1], we used the following method for the object identification (Figure 12). Suppose AVA$_i$ is tracking the target object and let L_i denote the 3D view line directed toward the object. Then, suppose AVA$_j$ detects an object and let L_j denote the 3D view line directed toward that object from AVA$_j$. AVA$_j$ reports L_j to AVA$_i$, which then examines the nearest 3D distance between L_i and L_j. If the distance is less than the threshold, a pair of detected objects by AVA$_i$ and AVA$_j$ are considered as the same object, and AVA$_i$ and AVA$_j$ forms what we call *agency*. The agency denotes a group of AVAs which are cooperatively tracking the same object. The agency consists of the *master* AVA and *worker* AVA(s): the master AVA is tracking the target object and maintains the object data. In the above example, AVA$_i$ is the master and AVA$_j$ a worker.

For the above mentioned object identification method to work properly, the following two conditions should be satisfied:

Geometric Calibration: 3D locations of multiple cameras should be well calibrated.

Temporal Synchronization: Image acquisitions by multiple cameras should be well synchronized.

While the geometric calibration can be done a priori

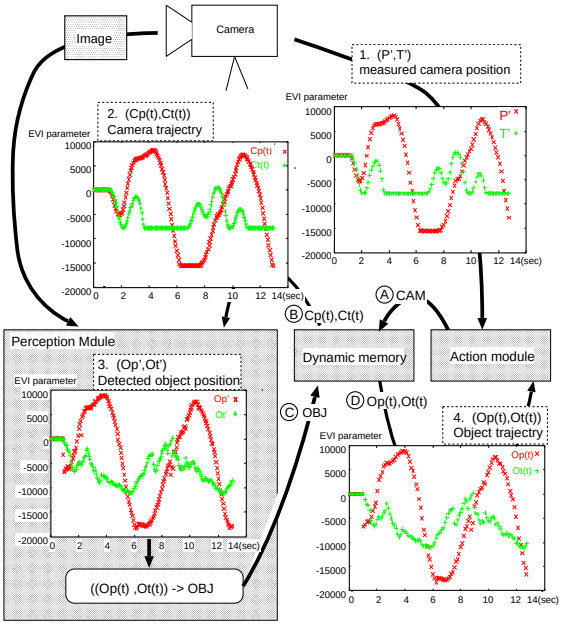


Figure 11: Dynamic data exchanged between the perception and action modules. Large amplitude: pan, Small amplitude: tilt. $(O_p(t),\ O_t(t))$ and $(C_p(t),\ C_t(t))$ in the figure denote object location $(P_{obj}(t),\ T_{obj}(t))$ and camera gaze direction $(P_{cam}(t),\ T_{cam}(t))$ in the text, respectively.

by off-line processing, the temporal synchronization should be attained on-line and has much to do with dynamics of AVAs. Assuming the geometric calibration has been done, here we will discuss the temporal synchronization among AVAs.

We could realize the synchronized image acquisition by the following methods:

Introduction of special hardware / software for synchronization: Although the synchronized image acquisition is guaranteed, the autonomy of each AVA is damaged; an AVA is forced to capture and process an image irrespectively of its own intention[2] .

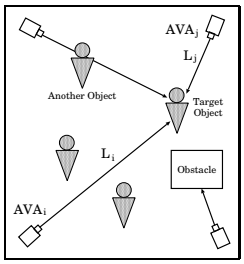


Figure 12: Cooperative gazing

[2] In [1], we used this method; the master AVA broadcasts a message for image acquisition to all worker AVAs.

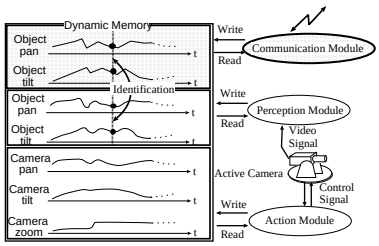


Figure 13: Virtual synchronization between multiple AVAs using the dynamic memory

Verification of the synchronization between observed images: In the above mentioned object identification method, for example, we can introduce time-stamps for L_i and L_j respectively. If the difference between the time-stamps is less than a certain threshold, we regard the image acquisitions as synchronized. In this method, while the autonomy of each AVA is preserved, the chance of the object identification decreases considerably; the object identification often fails because the image acquisition timing varies depending on the activity of each AVA.

What we want to realize is the synchronized observation among autonomous AVAs with asynchronous activities. With the dynamic memory, we can easily attain this rather conflicting goal.

Basically we employ the same object identification method as described before. The differences are as follows. (1) AVA$_i$ stores into its internal dynamic memory L_i associated with its observation timing. (2) AVA$_j$ reports to AVA$_i$ L_j associated with its observation timing. (3) Then, AVA$_i$ stores L_j into its dynamic memory to record the dynamics of the object tracked by AVA$_j$ (shaded graphs in Figure 13). (4) When AVA$_i$ wants to establish the object identification, it reads from the dynamic memory the values of L_i and L_j at the specified moment and computes the nearest 3D distance between the pair of 3D view lines based on the read-out values.

Thus, the dynamic memory enables the *virtual synchronization* between AVAs without disturbing their autonomous dynamics. Note that here also no waiting time is required for the synchronization and the timing of the object identification is determined solely by AVA$_i$.

6.2 Performance Evaluation

A computer-controlled mobile robot moves along an L-shaped track on the floor and four FV-PT cameras (i.e. AVAs) are placed 2m high above at the four corners of the room (Figure 14(a)). To introduce par-

	Spatial Error	Temporal Error
System 1	12.9518 (cm)	-
System 2	21.9445 (cm)	1.5593 (sec)

Table 1: Errors in the object identification.

tial view interferences from the cameras, a rather large box is placed at the center of the L-shaped track. The same communication protocol as used in [1] was employed for the cooperative tracking by communicating AVAs.

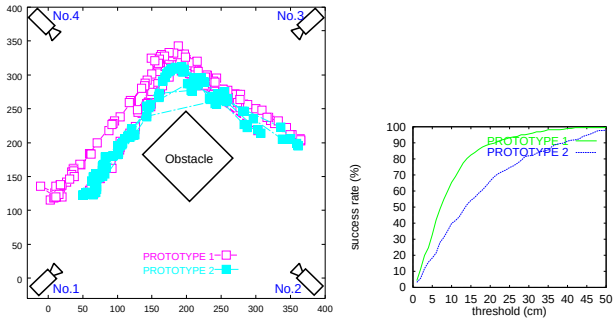
The white and black squares in Figure 14(a) illustrate object locations identified by Prototype system1 (with virtual synchronization) and Prototype system2 (without virtual synchronization) respectively. In both systems, when the nearest distance between 3D view lines is less than $\sqrt{500}$cm, the object is identified. In system1, when AVA_i observes a new object location, it tries to identify the object by reading L_i and L_j from the dynamic memory. In system2, on the other hand, when L_j is reported from AVA_j, AVA_i conducts the object identification using the most lately observed L_i. The spatial error in Table 1 denotes the average value of the nearest distance between L_i and L_j when the object identification is established. The temporal error for system2 denotes the average temporal difference between observation timings of L_i and L_j.

We can get the following observations:
(a) The virtual synchronization reduces the spatial error significantly, which hence increases the accuracy of the object localization.
(b) The above advantage is also verified in Figure 14(a): while the white squares in the figure are almost located on the object motion track, locations of the black squares deviate considerably from the track.
(c) We have more white squares than black ones in Figure 14(a), which implies the chance of the object identification is increased by the virtual synchronization.
(d) Figure 14(b) shows the success rate of the object identification when the threshold for the distance between L_i and L_j is changed. This also proves the effectiveness of the virtual synchronization.

7 Concluding Remarks

This paper proposes a novel dynamic system architecture named *Dynamic Memory Architecture*, where parallel processes and autonomous agents can interact asynchronously without disturbing their own intrinsic dynamics. The practical effectiveness of our idea has been demonstrated by the real time object tracking system using the dynamic memory.

This work was supported by the Research for the Future Program of the Japan Society for the Promotion of Science (JSPS-RFTF96P00501).



(a) (b)

Figure 14: Accuracy of the object identification. (a) Identified object locations (b) Rate of the object identification.

References

[1] Matsuyama, T.: "Cooperative Distributed Vision – Dynamic Integration of Visual Perception, Action, and Communication –," Proc. of Image Understanding Workshop, pp. 365-384, 1998

[2] Toyama, K. et al: "Wallflower: Principles and Practice of Background Maintenance," Proc. of ICCV, pp. 255-261, 1999

[3] Shafer, S.A., Stentz, A., and Thorpe, C.E.: "An Architecture for Sensor Fusion in a Mobile Robot," Proc. of IEEE Conf. on Robotics and Automation, pp.2002-2011, 1986

[4] Little, J.J. and Kam, J.: "A Smart Buffer for Tracking Using Motion Data," Proc. of Computer Architecture for Machine Perception, pp.257-266, 1993

[5] Zhang, Y. and Mackworth, A.: "Constraint Nets: A Sematic Model for a Hybrid Dynamic Systems," Theoretical Computer Science, Vol.138, pp.211-239, 1995

[6] Brown, C.M.: "Gaze Control with Interactions and Delays," IEEE Trans., Vol.SMC-20, No.1, pp.518-527, 1990