

ARGUS: An Automated Multi-Agent Visitor Identification System

Rahul Sukthankar*[†] and Robert G. Stockton*

*Just Research
4616 Henry Street
Pittsburgh, PA 15213

[†]The Robotics Institute
Carnegie Mellon Univ.
Pittsburgh, PA 15213

{rahuls,rgs}@justresearch.com

Abstract

ARGUS is a multi-agent visitor identification system distributed over several workstations. Human faces are extracted from security camera images by a neural-network-based face detector, and identified as frequent visitors by ARENA, a memory-based face recognition system. ARGUS then uses a messaging system to notify hosts that their guests have arrived. An interface agent enables users to submit feedback, which is immediately incorporated by ARENA to improve its face recognition performance. The ARGUS components were rapidly developed using JGram, an agent framework that is also detailed in this paper. JGram automatically converts high-level agent specifications into Java source code, and assembles complex tasks by composing individual agent services into a JGram pipeline. ARGUS has been operating successfully in an outdoor environment for several months.

Introduction

Consider the following scenario. Visitors to large apartment complexes are typically screened by a security guard in the lobby before being allowed to enter. Over time, guards learn to associate frequent visitors with the tenants whom they plan to visit, and are able to immediately notify the visitor's host of the guest's arrival over the building intercom. This paper presents an automated version of such a security guard: a multi-agent system for visitor identification, named ARGUS (after the vigilant watchman from Greek mythology).

At a high-level, ARGUS's operation consists of the following steps, each of which is managed by one or more JGram (Sukthankar, Brusseau, & Pelletier 1998) agents. A security camera photographs the building entrance every two seconds, and a motion detection algorithm identifies potential scenes containing visitors. Faces from these images are extracted using a neural-network-based face detector (Rowley, Baluja, & Kanade 1998). ARENA (Sim *et al.* 2000), a memory-based face recognition system, examines these face images and attempts to find visually similar matches in its stored

database of visitors. Users interested in receiving notification of visitors may run a JGram agent which will automatically be informed when the relevant visitors are identified. This agent also allows users to provide ARGUS with immediate corrections for identification errors. Since ARENA is capable of online learning, this feedback can be immediately incorporated into the recognition dataset.

ARGUS is implemented as a collection of agents in a multi-agent system (Sycara 1998) for several reasons. First, the components require different platforms: for instance, the camera interface is limited to Windows, while the face recognition system prefers Linux. Similarly, ARGUS users, distributed over an intranet, require notification on their individual workstations. (Some run Linux and others run Windows.) JGram agents, which use Java RMI for communication, are well suited for this scenario. Second, the computational load imposed by some of the image processing routines is severe enough to merit splitting the task over multiple machines. Third, a multi-agent architecture offers a high degree of modularity, allowing ARGUS agents to be dynamically added or removed from the system. For instance, interface agents can be created and killed as users arrive and leave without affecting the rest of the ARGUS system. Similarly, monitoring agents can be inserted to diagnose problems without disrupting service, and different face recognition algorithms can be seamlessly tested.

The remainder of this paper is organized as follows. First, the JGram agent framework is described. Next, the architecture of the ARGUS system is detailed, with an emphasis on the ARENA face recognizer. In the following section, ARGUS is evaluated in three experiments. The discussion explores strategies for improving visitor identification accuracy. The paper concludes by presenting promising directions for future research.

The JGram Agent Framework

The JGram agent framework was designed to simplify agent development in Java by providing strong support in three areas: (1) automatic generation of low-level agent code from high-level agent service specifications; (2) dynamic composition of agent services; (3) an in-

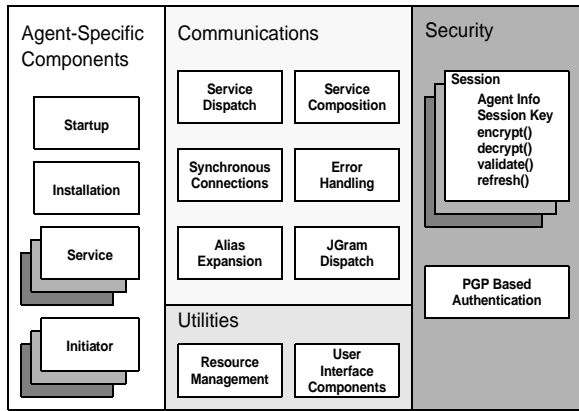


Figure 1: An overview of the internals of a JGram agent (See the text for details).

telligent scheme for handling errors (exceptions) across agents.

All inter-agent communications in the JGram framework is performed using an object known as a “slate”: essentially a small and self-contained blackboard that is passed from agent to agent, containing named objects that can be read, added or removed. By sending parameters to services as slates, JGram is able to multiplex multiple remote communications through a single remote method interface. All JGram agents can therefore provide reconfigurable agent interfaces while sharing a single, static, RMI stub file. Details are available in (Sukthankar, Brusseau, & Pelletier 1998).

Figure 1 shows the internals of a JGram agent. The JGram framework concentrates on providing general low-level aspects of agent communication. For instance, agent services can easily be executed in parallel, and can be invoked either synchronously (function call semantics) or asynchronously (message semantics). The JGram framework can also provide transparent support for secure interactions between agents: in the initial exchange, agents use public-key authentication and create a fast symmetric session-key for further communications. Note that the JGram framework is largely independent of higher-level issues such as agent communication languages or specific agent architectures. Rapid development of complex agent systems is possible because most of the components in the *Communications*, *Security* and *Utilities* sections (see Figure 1) are provided and Java source skeletons for the *Agent-Specific Components* are automatically generated from the high-level agent specification file. The JGram framework also provides a name server for alias expansion, managing lists of agents and public key dissemination.

A variety of Java-based agent frameworks have recently become available (Jeon 1998; Chauhan & Baker 1998). JGram differs from these in that it does not provide agent templates nor support for specific agent

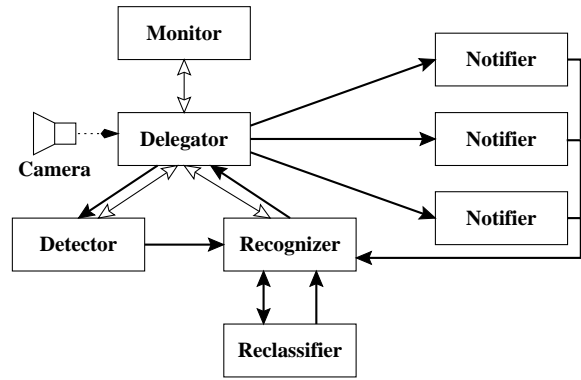


Figure 2: This diagram shows an overview of the ARGUS architecture, where each box depicts a JGram agent. The heavier lines show the major data pathways and the light lines show monitoring information. A line with a double arrows represents a synchronous exchange while one with a single arrow indicates asynchronous dataflow. (See the text for details.)

methodologies. Instead, JGram enables rapid development of communities where the agent interactions can be modeled as service requests. Additionally, JGram provides a novel, cross-agent generalization of pipes and exceptions that significantly simplifies development of certain complex applications. See (Sukthankar, Brusseau, & Pelletier 1998) for details.

An Overview of ARGUS Components

This section presents an overview of the ARGUS system architecture (see Figure 2) and details the important agents and their interactions. ARGUS’ three tasks may be summarized as: (1) visitor identification and notification; (2) interactive labelling of unknown visitors; (3) evaluation of face detection and face recognition algorithms. Since these tasks occur in parallel, most ARGUS agents perform several roles simultaneously.

In the primary task, visitor identification, the *delegator* agent collects images from the camera hardware (every 2 seconds), performs image differencing (Ballard & Brown 1982) on successive images, and sends those images that contain motion through a JGram pipeline to the *detector* and *recognizer* agents. In the event of a positive visitor identification, the *delegator* broadcasts messages to all *notifier* agents that are interested in this visitor’s arrival. User feedback acquired from the *notifier* agents is used by the *recognizer* to update its face database.

The secondary task, labelling unknown visitors, is performed every few days by ARGUS administrators, through the *reclassifier* interface agent. The *reclassifier* requests a list of unlabelled faces from the *recognizer* and allows the administrator to add them as training examples. The *reclassifier* agent is also used to correct

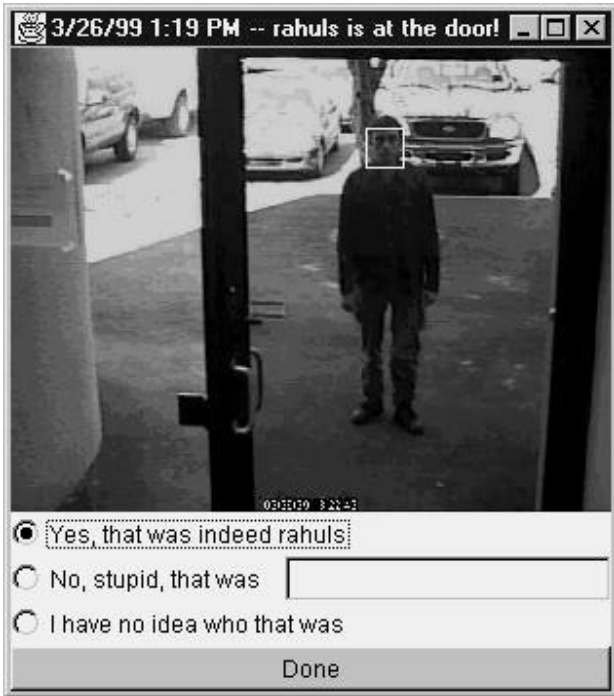


Figure 3: The ARGUS notifier displays an image captured by the security camera along with a box surrounding the face of the visitor and a tentative identification. User feedback is used to improve face recognition. (The window size was reduced for publication.)

errors in the face database, and to dynamically create new visitor classes. In the current implementation, the *recognizer* does not require offline training, so the updates are immediately available for the primary task.

ARGUS' final task is supported by the *monitor* agent that allows users to interactively query evaluation metrics collected by all of the other ARGUS agents. This is particularly useful when the *recognizer* agent is evaluating experimental algorithms in addition to using an established algorithm for the primary task. In addition, the *monitor* simplifies remote administration of the visitor identification system by verifying that all agents are operating normally.

Since we cannot present all of these agents in detail due to space limitations, we restrict ourselves to a brief discussion of the highlights. The *delegator*, in addition to its image processing responsibilities, also operates as a facilitator for the agent community, forwarding messages between different agents. The *detector* uses the neural-network-based face finder described in (Rowley, Baluja, & Kanade 1998). Its primary limitation, that it can only reliably detect upright, frontal faces, is not a major problem for ARGUS, since visitors typically face forward as they walk through the building entrance. The neural network weights were trained to locate faces

from web photographs and are not specialized for the visitor identification task. Since the face detector locates very tightly cropped faces (typically without hair and ears), ARGUS enlarges the recommended region by 140% in each dimension before extracting the face image. A *notifier* agent runs on each ARGUS user's personal workstation and pops up a window with an image of the visitor (as shown in Figure 3). The user may restrict notification to a subset of visitors, and *notifier* agents may join and leave the ARGUS community at any time. The *recognizer* employs a novel algorithm and is fully described below.

Face Recognition

The visitor identification problem is well served by a memory-based face recognition algorithm for the following reasons. First, since the set of visitors is not known *a priori*, the algorithm should be able to accommodate such changes easily. Memory-based algorithms (Atkeson, Moore, & Schaal 1997) do not have an explicit training phase and are ideally suited for incremental training. Second, the training data should be acquired without asking the visitors to pose for photographs under controlled conditions. A collection of several images per visitor, spanning a variety of illumination conditions can be used instead. Fortunately, non-parametric approaches, such as nearest-neighbor matching, may perform well even if these images do not form a single cluster in feature space. Finally, since the face recognition is used only for notification rather than access control, the cost for mis-identification is not severe. Thus, poor initial accuracy on a new visitor may be tolerated by users since recognition accuracy improves as feedback is provided.

The current ARGUS *recognizer* agent uses ARENA, a view-based face recognition system. The training phase is summarized as follows: (1) Minimize illumination variations using histogram equalization (Ballard & Brown 1982); (2) From each training image, generate 10 additional synthetic training images by making small, random perturbations to the original (rotation upto $\pm 5^\circ$, scale upto $\pm 5\%$, and translation upto ± 2 pixels in each direction). (3) Create reduced-resolution (16×16) versions of these images (using simple averaging over rectangular regions) and store them into the face database. Our experiments (see below) indicate that classification speed may be increased by a factor of 11, in exchange for a 5% drop in accuracy, by omitting step (2) when the database contains a large selection of images for each visitor.

The classification phase is similarly straightforward: (1) Pre-process the input image using histogram equalization; (2) Create a reduced-resolution image as discussed above; (3) Return the label of the single nearest-neighbor to the input image, among the stored low-resolution images, using the $L_{0.5}$ metric.¹ (As shown

¹ $L_{0.5}(\vec{a} - \vec{b}) \equiv \left(\sum \sqrt{|a_i - b_i|} \right)^2$.



Figure 4: Top row: Sample images of two people, as extracted by the face detector. The image quality is poor due to lighting and camera placement. Also note the variation in appearance due to differences in illumination, pose and facial expression. Bottom row: the corresponding ARENA reduced-resolution images (16×16) pixels.

in (Sim *et al.* 2000), the Euclidean metric, L_2 , does not perform as well.) Figure 4 shows sample faces extracted from images by the face detector, along with the corresponding (enlarged) ARENA reduced-resolution images.

Although face recognition is an old problem, it has received much attention in the last few years (Chellappa, Wilson, & Sirohey 1995; Fromherz 1998). The research effort has largely focused on the subproblem of frontal face recognition, and in this domain, techniques based on Principal Components Analysis, popularly termed *eigenfaces* (Turk & Pentland 1991; Pentland, Moghaddam, & Starner 1994) have demonstrated good performance. Most published results report experiments on standard datasets such as ORL (Samaria & Harter 1994) or FERET (Phillips *et al.* 1997).

Our extensive experiments with ARENA and PCA-based methods on both FERET and ORL datasets appear in (Sim *et al.* 2000). Here, we present only the results of one such test: Table 1 summarizes a direct comparison of ARENA with the best published face recognition results (Lawrence *et al.* 1996) on the ORL database. The ORL database contains 10 frontal images of each of 40 people taken under consistent conditions. The three columns in the table indicate how many of these images were placed in the training set (the rest were used for testing). “CN” and “SOM” refer to convolutional neural network and self-organizing map respectively; see (Lawrence *et al.* 1996) for details. One could reasonably argue that a simple system like ARENA achieved this accuracy solely by exploiting the relatively uniform illumination and background conditions present in the standard datasets. Thus, one of our primary motivations for integrating ARENA in ARGUS was to evaluate its accuracy in a more challenging, real-world setting.

| Images per person | 1 | 3 | 5 |
|-------------------------|--------------|--------------|--------------|
| Eigenface avg per class | 61.4% | 71.1% | 74.0% |
| Eigenface one per img | 61.4% | 81.8% | 89.5% |
| PCA+CN | 65.8% | 76.8% | 92.5% |
| SOM+CN | 70.0% | 88.2% | 96.5% |
| ARENA $L_{0.5}$ | 76.2% | 92.7% | 97.4% |

Table 1: Comparison of ARENA with results reported in (Lawrence *et al.* 1996) on the ORL dataset.

Experiments in Visitor Identification

This section summarizes three experiments that evaluate ARGUS on the task of visitor identification. In accordance with the terminology used in the FERET (Phillips *et al.* 1997) test, the set of labeled training images is termed the *gallery* and a test face (which must not appear in the gallery) is termed a *probe*. All of the experiments reported here used face images collected by ARGUS between January and March 1999. Compared to the standard datasets used in the face recognition community, these are very challenging photographs for the following reasons: The images were taken by an outdoor camera in a variety of lighting conditions, at different times during the day, and in very different weather conditions (including snow and rain). Since visitors were not instructed to pose for the camera, their head orientations varied significantly, and many of the individuals were wearing accessories such as hats, hoods or sunglasses. Several of the night images resulted from internal reflections of individuals as they approached the glass doors from the interior of the building. The extracted face image sizes ranged from 33×33 to 100×100 pixels, with a median face image size of 47×47 .

Leave-One-Out Tests

In the Leave-One-Out (LOO) tests, each of the faces in the gallery was used as a probe image. The probe and its synthesized images were temporarily removed from the gallery, and ARGUS was asked to identify the individual in the probe. The fraction of probes that were correctly classified is reported as the accuracy. Two versions of this experiment were conducted: one with almost all of the stored faces (973 images from 44 individuals), and the second restricted to photographs of the most common visitors (881 images from 23 individuals). On the first version, ARGUS achieved an overall classification accuracy of 64.1% (60.4% without synthetic images). The second version of this test focused on the task of identifying regular visitors. ARGUS correctly identified 69.7% of probes in this test (65.2% without synthetic images). Two individuals (shown in Figure 4), each with approximately 100 images in the gallery, were correctly identified approximately 90% of the time.

Online Training

The LOO tests described above model the visitor identification task inaccurately in one important respect:

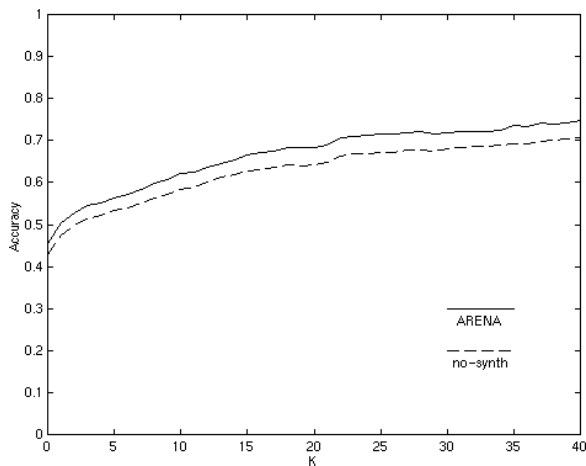


Figure 5: This graph illustrates the variation in face recognition accuracy with respect to k , the minimum number of raw training images per individual present in the database. The dashed line shows accuracy for the version of ARENA without synthesized training data.

when multiple images of a person, all taken within a short span of time, are present in the gallery, a probe selected from this subset is likely to be very similar to a labeled image in the gallery. However, in a real visitor identification task, all of these images would have appeared together as probes and none of the probe images would be eligible to match the other (visually similar) images in its batch. The Online Training experiment addressed this potential criticism by more faithfully simulating the sequence of online training that occurs in ARGUS.

The gallery was initialized to be the empty set. The stored images were successively fed (as probes) to ARGUS in time-stamp order. During the face recognition step, the matching was restricted to images that had been acquired at least five minutes earlier than the probe image’s timestamp. This prevented ARENA from exploiting any image similarity that may have existed among images taken at approximately the same time.

The Online Training experiment explored how recognition accuracy varies with k , the minimum number of raw training images (per individual) present in the database (see Figure 5). For example, when $k = 20$, only the 11 most common visitors (each with at least 20 images in the database) are retained in the gallery. Recognition accuracy improves as k increases, showing that ARGUS is fairly accurate over a limited subset of visitors. The second line shows that, without synthetic images, recognition accuracy lags by 3%–5%.

Robustness Tests

ARGUS has been operational at Just Research for the last two months. To increase speed, we have eliminated the synthesized images from the ARENA training set. The current system is quite responsive: the *notifier* typically displays a window within seconds of a visitor’s arrival, and occasionally before the visitor has even pressed the doorbell.

The observed recognition accuracy for common visitors (individuals with 10 or more training images) was 53.9%. For the full case, even including visitors for which no training data exists, the observed accuracy was 43.4%.

To explore ARGUS’ resilience to distractors, approximately 1500 spurious images of faces collected by a web spider (all labeled as “stranger”), were added to the database of 750 actual camera images. This change only reduced the observed accuracy by 3.1%. Although new visitors and infrequent visitors, with few examples in the database were often misclassified as “stranger”, the distractors had surprisingly little detrimental impact on the regular visitor (or overall) recognition accuracy. This strengthens the hypothesis that ARENA’s simple view-based nearest-neighbor classification scheme may be more robust than anticipated.

Discussion

Although ARENA achieves excellent results on the standard datasets, its performance on the ARGUS data shows that there is clearly room for improvement. Strategies for improving accuracy can be divided into three classes: higher-quality input data; combining multiple classifiers; and better face recognition techniques.

The current security camera images used for ARGUS (See Figures 3 and 4) are fairly poor-quality for several reasons. First, the visitor only occupies a small area in a wide field of view. Second, during daylight hours, images tends to be severely back-lit because the camera is looking from relative darkness, through a glass door, into direct sunlight. At night, the tinted glass acts as a (partial) mirror, causing reflections of the building interior to appear over visitors’ faces. Third, since visitors are not asked to pose for the camera, they may appear anywhere in the image, with their faces oriented suboptimally. In fact, many of the images are of partially-illuminated faces that are difficult for humans to identify.² Thus, improving camera position, installing controlled lighting and asking visitors to pose would all improve ARGUS’ recognition accuracy (and make the problem less interesting from a research standpoint).

Accuracy could also be improved by combining the outputs from multiple, independent face recognition algorithms using voting. Alternatively, ARGUS could

²Although they have difficulty identifying the visitor from the face alone, humans are able to make intelligent guesses based on clothing, overall size and time of day.

wait for consistent classification of several successive images before issuing a notification. Either of these schemes offer potential for user customization.

We have also experimented with several variants of ARENA that show promising performance on this task. These include: (1) better preprocessing of input images to compensate for lighting conditions; (2) simple feature extraction using wavelet decomposition; and (3) incorporating texture information using Gabor filters. The ARGUS architecture enables us to evaluate several recognition algorithms in parallel and transparently upgrade the visitor identification system without interruption of service.

Conclusions and Future Work

ARGUS solves a real-world application by combining image processing, machine learning and messaging technologies in a multi-agent framework. However there is still work to be done:

- While we have evaluated the individual components in ARGUS, we plan to get a better idea of the overall system's performance by getting feedback from a larger population of users. We are also interested in observing how the performance of a memory-based face recognizer scales with large populations of visitors.
- Since it is collecting a large dataset of labeled human faces, with multiple images taken over a period of time, ARGUS enables us to easily test different face recognition systems in real-world situations. This dataset may be valuable to others in the face recognition community, and we hope to make it publicly available over the web.
- The ARGUS implementation easily accommodates the addition of new types of agents. For instance, the visitor could be automatically notified if his/her host were unavailable, by an interface agent at the building entrance.

Acknowledgments

The JGram agent system was developed in collaboration with Antoine Brusseau and Ray Pelletier. The ARENA face recognition system was developed in collaboration with Terence Sim, Shumeet Baluja and Matt Mullin. The face detection software was provided by Henry Rowley. Thanks to Gita Sukthankar and Martha Underwood for their valuable feedback on this paper.

References

- Atkeson, C.; Moore, W.; and Schaal, S. 1997. Locally weighted learning. *AI Review* 11.
- Ballard, D., and Brown, C. 1982. *Computer Vision*. Prentice-Hall.
- Chauhan, D., and Baker, A. 1998. JAFMAS: A multi-agent application development system. In *Proceedings of Autonomous Agents*.
- Chellappa, R.; Wilson, C.; and Sirohey, S. 1995. Human and machine recognition of faces: A survey. *Proceedings of the IEEE* 83(5).
- Fromherz, T. 1998. Face recognition: A summary of 1995-1997. Technical Report ICSI TR-98-027, International Computer Science Institute, Berkeley.
- Jeon, H. 1998. An introduction to JATLite. Technical report, CDR, Stanford University. <http://java.stanford.edu/java/_agent/html/>.
- Lawrence, S.; Giles, C.; Tsoi, A.; and Back, A. 1996. Face recognition: A hybrid neural network approach. Technical Report UMIACS-TR-96-16, University of Maryland.
- Pentland, A.; Moghaddam, B.; and Starner, T. 1994. View-based and modular eigenspaces for face recognition. In *Proceedings of Computer Vision and Pattern Recognition*.
- Phillips, P.; Moon, H.; Rauss, P.; and Rizvi, S. 1997. The FERET september 1996 database and evaluation procedure. In *Proceedings of Audio and Video-based Biometric Person Authentication*.
- Rowley, H.; Baluja, S.; and Kanade, T. 1998. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(1).
- Samaria, F., and Harter, A. 1994. Parametrisation of a stochastic model for human face identification. In *Proceedings of IEEE Workshop on Applications on Computer Vision*. ORL database is available at: <www.cam-orl.co.uk/facedatabase.html>.
- Sim, T.; Sukthankar, R.; Mullin, M.; and Baluja, S. 2000. Memory-based face recognition for visitor identification. In *Proceedings of Face and Gesture*.
- Sukthankar, R.; Brusseau, A.; and Pelletier, R. 1998. A gentle introduction to developing JGram agents. Technical report, Just Research.
- Sycara, K. 1998. Multiagent systems. *AAAI AI Magazine* 19(2).
- Turk, M., and Pentland, A. 1991. Eigenfaces for recognition. *Journal of Cognitive Neuroscience* 3(1).