ARPA Image Understanding Workshop
Nov 1994

Software Library for Appearance Matching (SLAM)

Sameer A. Nene, Shree K. Nayar and Hiroshi Murase
Center for Research in Intelligent Systems, Department of Computer Science,
Columbia University, New York, N.Y. 10027, U.S.A.

ABSTRACT

The SLAM software package has been developed for appearance learning and matching problems in computational vision. Appearance learning involves use of principal component analysis for compression of a large input image set to a compact low-dimensional subspace, called the eigenspace, in which the images reside as parameterized manifolds. SLAM enables the user to obtain this parametric representation by providing modules for eigenspace computation, projection of images to eigenspace, and interpolation of multivariate manifolds through the projections. Appearance matching is done by searching for a projection in eigenspace closest to a novel input projection. Algorithms have been provided for performing this search in real-time, even with huge datasets. Benchmarks demonstrate the suitability of SLAM for application to real-world problems. The functionality has been made available to the user through an X/Motif Graphical User Interface along with command-line programs and a C++ class library. Use of object oriented techniques provides an easy to use and extensible Application Programming Interface.

1 INTRODUCTION

Computer vision algorithms often use simple template matching techniques for object recognition or feature detection. While this works well in cases where matching is performed on a small number of templates, it can cause severe computational and memory problems when it is desired to recognize or match a large number of images. Image compression techniques, like *principal component analysis* [Oja 1983], partly alleviate the problem. This method computes eigenvectors of an image set, which form an orthogonal basis for representing individual images in the set. Though a large number of eigenvectors may be required for accurate reconstruction of an image, only a few eigenvectors with the highest eigenvalues are generally sufficient to capture the significant appearance characteristics. This method has been previously applied to edge detection [Hummel 1979] and human face recognition [Turk and Pentland 1991]. Although principal component analysis improves on the computational requirements, it is not as successful in cases where accurate matching is required for incremental changes in appearance. This is the case when, for example, an object has to be recognized in all orientations (poses). In order to achieve reasonable pose estimation accuracy, one would need to have an impracticably large input image set.

Murase and Nayar propose a novel parametric representation that captures appearance of complex 3D objects in various poses and illumination conditions [Murase and Nayar 1993]. The representation is succinct and its computation does not require a large number of input images. We will briefly describe their approach. The first step, *visual learning*, is to compute eigenvectors of the input image set. As mentioned earlier, the eigenvectors form an orthogonal basis for representation of the images. Since only a small number of eigenvectors contribute to the appearance, the basis can be truncated to obtain a smaller subspace, called *the eigenspace*. The images are then projected to eigenspace to obtain a set of discrete points. Interpolation between these points obtains a manifold, which is a parametric representation of the input image set.

This parametric representation can be used for efficient appearance matching. The basic methodology used for matching is as follows. An input image is projected to eigenspace to obtain a single point. A search is then carried out to find a point on the manifold closest to the projected point. This manifold point represents the best match.

The above processes of learning and matching have been effectively used by Murase and Nayar to implement an object recognition and pose estimation system which recognizes 20 complex 3D objects in real-time [Murase and Nayar 1994b]. In a related paper, the parametric eigenspace representation is used in planning illumination for object recognition [Murase and Nayar 1994a]. Nayar *et al.* use appearance learning and matching to position a

robotic manipulator based on raw brightness images from an uncalibrated camera [Nayar *et al.* 1994a, 1994b]. In the same work, they also demonstrate visual tracking of moving objects and defect inspection of manufactured products.

SLAM closely follows the approach described in all the above work while attempting to maximize scope and generality. We were motivated to develop SLAM because we view the parametric eigenspace technique as a generic tool for appearance learning and matching problems in computer vision.

2 LEARNING APPEARANCE

This section describes the tools available in SLAM for learning appearance. Learning involves preparation of vector sets, computation of eigenvectors (eigenspace), projection of images to eigenspace, and interpolation of a manifold through the projected points. All the above operations can be done with the help of a X/Motif graphical user interface or command-line programs.

2.1 Preparation of Feature Sets

The first step is to prepare a set of vectors that can be used for computing eigenvectors. Unprocessed brightness images, or processed images such as smoothed images, first derivatives, Laplacian, power spectrum of the brightness images, or any combination of such images, can be used as vectors [Murase and Nayar 94b]. A collection of such vectors, possibly related by some common appearance characteristic, is called a *vector set.* Since vector sets are not necessarily "image" sets, they are also called *feature sets.*

SLAM provides an image manipulation module that can be used for conversion of image sets to vector sets. A processing step, such as brightness normalization, segmentation, size normalization or any of their combinations may be involved during conversion to vectors [Murase and Nayar 1993]. Furthermore, this process would typically involve conversion and processing of hundreds of images. The image manipulation module has facilities to perform all the above and in addition, provide visualization capability (see Fig. 1). This is especially useful as it lets the user view images, vectors, eigenvectors and has a simple VCR like interface to animate image sequences. It also gives immediate visual feedback to the user on how various processing options would affect the data sets.

A typical session with the image manipulation module might be as follows: The user selects the Open option from the File menu to pop up a dialog box. This dialog box will allow the user to load images, vectors, image sets, vector sets or eigenvectors. After the user loads the

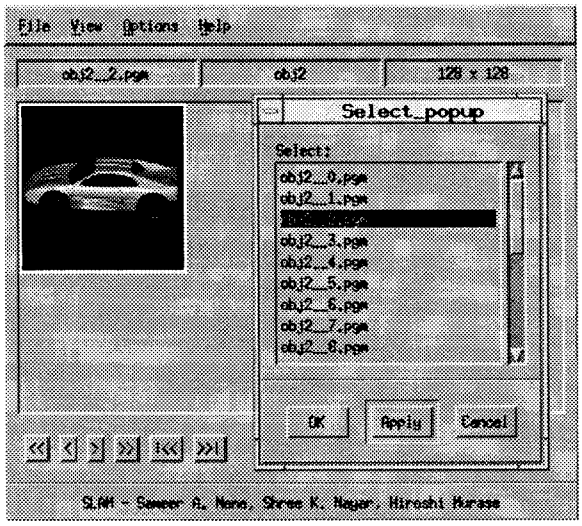

Fig 1: Screen shot of Image Manipulation Module

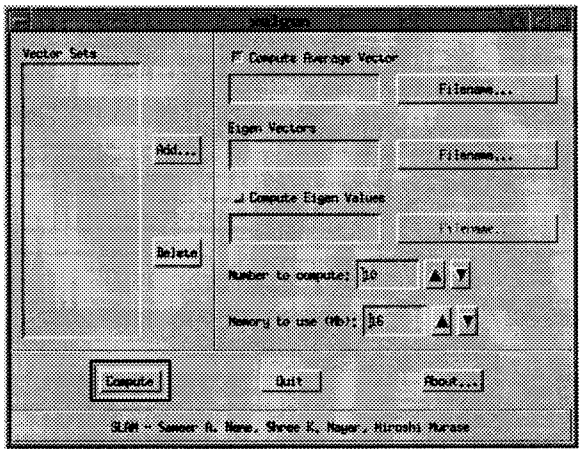

Fig 2: User Interface to compute Eigenspace

desired entity, say an image set, he/she might use the VCR interface to inspect the image set. Then, he/she could select one or more processing options from the options menu and get immediate feedback on how they would affect the entire image set. If he/she is not satisfied, he/she could, for example, fine tune the segmentation threshold or the normalization size using the Preferences dialog. Finally, he/she would want to convert the entire image set to a vector set by going to the Save As dialog in the File menu and selecting the option Vector Set.

2.2 Subspace Computation

Once a feature set has been prepared, the stage has been set for computing an optimal subspace (or eigenspace) for that set. SLAM provides two programs for this purpose, an X/Motif user interface (see Fig. 2), and a command-line program for the experienced user. We mentioned earlier that the eigenspace is usually trun-

cated, *i.e.* it only contains eigenvectors with the highest eigenvalues [Oja 1983]. It is very difficult to predetermine how many eigenvectors contribute significantly to the appearance. For this reason, SLAM also computes eigenvalues, which are often used as a guideline to help decide the truncation point. A universal eigenspace [Murase and Nayar 1993] can be computed by simply specifying multiple vector sets.

A typical session with the X/Motif interface might involve specifying the vector sets through a dialog, setting the number of eigenvectors to compute, supplying the filenames for the average vector, eigenvalues, and eigenvectors, setting the amount of memory to be used, and finally starting the actual computation. For the advanced user, the same functionality is available via the command-line.

We use a time efficient implementation (see Table 1) of an algorithm based on singular value decomposition [Murakami and Kumar 1982] along with a conjugate gradient algorithm [Yang *et al.* 1989] for computation of eigenvectors. The reader will appreciate that the performance is very good inspite of the large sizes of the input vector sets.

Architecture	Time	
	81 images	1440 images
Sun Sparc IPX	132 sec.	442 min.
DEC Alpha 3600	26 sec.	78 min.
HP PA 9000/735	68 sec.	118 min.
SGI Onyx[a]	36 sec.	91 min.

Table 1: Time taken to compute 20 eigenvectors from images of size 128 x 128.

a. With disk striping.

2.3 Parametric Eigenspace Representation

Once an eigenspace has been computed, the input vector set has to be projected to this space to obtain discrete high-dimensional points. By interpolating between these points, one can obtain a manifold that forms a parametric representation of the input vector set. We use a quadratic BSpline for the interpolation [Rogers 1990]. This BSpline (manifold) can be (re)sampled at a higher frequency to obtain a dense set of discrete points. As we shall see in the next section, these points can be used for appearance matching. It is possible that a user has computed manifolds for a number of vector sets, all *in* separate eigenspaces. If it is desired to contain all these manifolds in the same space, one need not always recompute a new eigenspace. This is possible by use of Gram-Schmidt orthogonalization [Householder 1964], which computes a space orthogonal to two or more (input) eigenspaces. All the operations we described above (projection, interpolation, resampling and orthogonalization) can be carried out with the help of SLAM modules. An X/Motif graphical interface (see Fig. 3) allows the user to visualize the projections and manifolds.

We now describe a typical interactive session with the Motif based module. We assume that a user needs to construct a parametric representation for a vector set whose eigenvectors have been already computed. The user first needs to project the input vectors to eigenspace. This can be done by the Project option in the Options menu. This dialog box lets the user specify the vector set to be projected, the eigenspace to use for projection, (optionally) the average vector, and the name of the output projection. Once the user fills in the required information, the vector set is projected to eigenspace. The projections are immediately visible on the screen as points inside a cube representing the eigenspace axes. The eigenspace has typically more than three dimensions. The program lets the user select which of the three dimensions should be visible with the help of the Preferences dialog. The user can interactively pan around, zoom or pan over the cube with the help of the mouse to select an appropriate view. The axes are labeled to denote which three dimensions are visible. At this point the user might want to save the projections to disk and/or obtain a hard (PostScript) copy of the current view.

The discrete points obtained above can be interpolated to obtain a BSpline with the help of the Interpolate option from the Option menu. Interpolation is carried out after selecting the projection to be interpolated and naming the output BSpline. The BSpline can be a curve, a surface or a volume. Let us assume in this case that the user interpolates a BSpline surface. The surface will then be immediately seen on the screen along with the previous projections. As earlier, it is possible to select an appropriate view or select which three eigenspace dimensions are visible. BSplines are displayed as series of short line segments by sampling them at a user-configurable frequency. The frequency determines how "smooth" the BSplines will look. If a BSpline were a volume, it would be still visible as a surface. In this case, the user would be able to select which two parameters are to be used to obtain the viewable surface and the fixed value of the third parameter. After the user is satisfied with the view, he/she may choose to save the manifold to disk and/or to obtain a PostScript copy.

3 APPEARANCE MATCHING

The final step is to use the parametric eigenspace representation obtained above for appearance matching. This

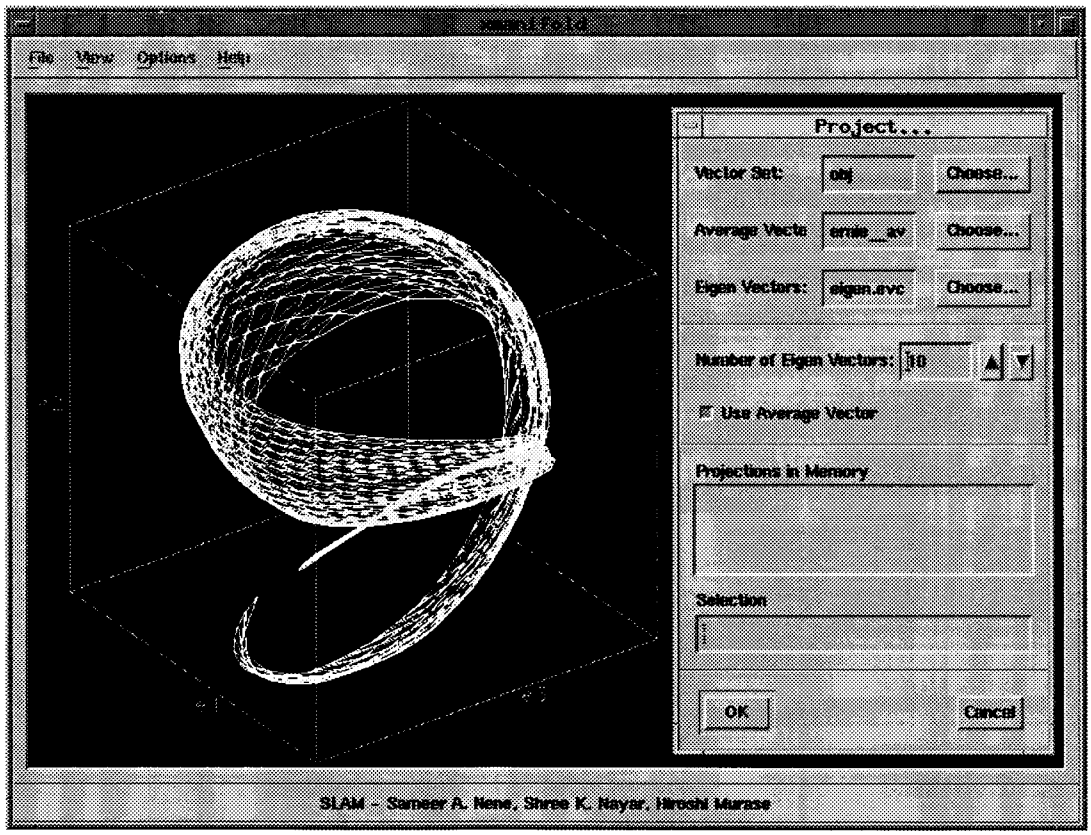

Fig 3: Manifold Manipulation Module

is done by projecting a novel image to eigenspace and finding a point on the manifold closest to that projection [Murase and Nayar 1994b]. In general, analytically finding the closest point on a high-dimensional manifold is a nontrivial problem. It is somewhat simplified if we rediscretize the manifold to obtain a dense set of points and search for the closest point within this set.

As described in the previous section, SLAM lets the user resample the manifold at any desired frequency to obtain a discrete point set. The user may choose to implement his own search or use a set of search algorithms offered by SLAM. Before we describe these algorithms, we must mention that they are available through a C++ Application Programming Interface (API) as *methods*. We chose not to develop a Motif interface for searching because of the considerable difficulties involved in programming and using low-level software (device drivers) and hardware (digitizers) consistently across diverse user environments.

3.1 Exhaustive Search

SLAM implements a brute-force search algorithm which goes through all the points in the resampled point set, computes the Euclidean distance between each of these points and the input projected point, and concludes the point with the smallest distance value to be the closest (best match). Although this method is simple and guaranteed to work, it is often impractical from a computational viewpoint because it involves calculating the Euclidean distance from possibly hundreds of thousands of high-dimensional points.

An alternative search algorithm implemented in SLAM uses a simple heuristic to reduce the number of points searched. This is done by considering only points that are sufficiently close to the input point in the first dimension. Since the points have maximum spread in the first dimension, a large number of points will be struck off the list of points to which the Euclidean distance has to be computed. This method unfortunately requires a threshold to determine how close is "close". If it is possible to have a rough idea about the threshold in advance, then this method offers better performance.

3.2 Binary Search

Both the schemes described above are not very useful when it is desired to search through a large number of points in real-time. Nene and Nayar propose a high-dimensional binary search algorithm to significantly improve performance from $O(kn)$ to $O(log2(kn))$ where k is the number of dimensions and n the number of points [Nene and Nayar 1994]. The algorithm partitions the space into its constituent dimensions and carries out a

binary search separately in each dimension. The closest point is obtained by looking at the overlap in all dimensions. SLAM includes an efficient implementation of this algorithm. We will not describe the algorithm in more detail here, but only mention that it also requires the use of an appropriately chosen threshold. See Table 2 for a benchmark of the three search algorithms we have described.

Search Algorithm	Time (ms.)	
	DEC Alpha 3600	Sun SPARC IPX
Exhaustive	26 (79)	121 (480)
Heuristic	15 (68)	32 (389)
Binary	6 (54)	8 (370)

Table 2: Time taken to search for the closest point from 7220 data points in 15-D eigenspace. The figures in parentheses indicate the time for projection *and* search.

4 OBJECT ORIENTED DESIGN

We chose to develop SLAM in C++ to give the user the ease and versatility of object oriented design (OOD). Consequently the SLAM Application Programming Interface (API) is also C++ based and is essentially a library of classes/methods. Although the user does not usually need to concern himself/herself with the API, it might be sometimes necessary to do so, for example, in case the user wishes to utilize the search algorithms.

SLAM uses advanced OOD concepts [Booch 1994] to a great extent in implementation of all the algorithms and procedures described above. Protocol (virtual) classes are used consistently to attain a consistent look and feel across the wide variety of classes and entities the user has to handle. For instance, disk based I/O is done with the help of a Persistent protocol class, which makes it possible to issue simple one line commands to load/store any of the complex entities (Vectors, Manifolds, Projections, Search databases, etc.). Similarly, it is possible to deal with any manifold (BSpline curves, surfaces, volumes, etc.) in a consistent fashion with the help of the Interpolation protocol. For searching, the SearchScheme protocol makes it possible to write algorithm independent application code that does not have to worry about the underlying search algorithms. The protocol classes not only help in writing "clean" code, but also support easy addition of user-defined functionality. For instance, if the user wishes to perform interpolation using wavelets, he/she simply has to write code which conforms to the Interpolation protocol and link the object(.o) file to the manifold manipulation module to get the full functionality of the graphical interface.

5 SLAM DISTRIBUTION

Columbia University is currently in the process of licensing SLAM to a number of educational institutions and industrial organizations. For information on obtaining SLAM, the reader is advised to contact Shree K. Nayar, Computer Science Dept., Columbia University, New York, NY 10027 or send email to *slam@cs.columbia.edu.*

SLAM works with most major workstation architectures (SunOS, Solaris, OSF/1, HPUX, Irix) and is available as executable binaries and/or source code. For information on availability for other (currently unsupported) architectures such as Ultrix, AIX, MS-DOS, etc., the reader is again advised to send mail to *slam@cs.columbia.edu.*

6 REFERENCES

[Booch, 1994] G. Booch, Object oriented analysis and design, 2nd ed., Benjamin/Cummings, CA, 1994.

[Householder, 1964] A. S. Householder, *The theory of matrices in numerical analysis,* Dover Publ., New York, 1964.

[Hummel, 1979] R. A. Hummel, Feature detection using basis functions, *Computer Graphics and Image Processing,* Vol. 9, pp. 40-55, 1979.

[Murakami and Kumar, 1982] H. Murakami and V. Kumar. Efficient calculation of primary images from a set of images, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 4(5):511-515, September 1982.

[Murase and Nayar, 1993] H. Murase and S. K. Nayar. Learning and recognition of 3d objects from appearance, *IEEE Qualitative Vision Workshop, CVPR,* New York, June 93.

[Murase and Nayar, 1994a] H. Murase and S. K. Nayar. Illumination Planning for object recognition in structured environments. *IEEE Conference on Computer Vision and Pattern Recognition,* June 1994.

[Murase and Nayar, 1994b] H. Murase and S. K. Nayar. Visual learning and recognition of 3d objects from appearance. *Intl. Journal of Computer Vision,* 1994. Accepted.

[Nayar *et al.*, 1994a] S. K. Nayar, H. Murase and S. A. Nene. Learning, positioning, and tracking visual appearance. *IEEE Intl. Conf. on Robotics and Automation,* May 1994.

[Nayar *et al.*, 1994b] S. K. Nayar, H. Murase and S. A. Nene. General learning algorithm for robot vision. *Proc. of ARPA Image Understanding Workshop,* Monterey, Nov. 94.

[Nene and Nayar, 1994] S. A. Nene and S. K. Nayar. Binary search through multiple dimensions. Technical Report CUCS-018-94, Department of Computer Science, Columbia University, New York, NY, USA, January, 1994.

[Oja, 1983] E. Oja, *Subspace methods of pattern recognition,* Research Studies Press, Hertfordshire, 1983.

[Rogers, 1990] D. F. Rogers, *Mathematical elements for computer graphics,* 2nd ed., McGraw-Hill, New York, 1990.

[Turk and Pentland, 1991] M. A. Turk and A. P. Pentland, Face recognition using eigenfaces. *Proc. of IEEE Conf. on Comp. Vision and Pattern Recog.,* pp. 586-591, June 1991.

[Yang *et al.*, 1989] X. Yang, T. K. Sarkar, and E. Arvas, A survey of conjugate gradient algorithms for solution of extreme eigen-problems of a symmetric matrix. *IEEE Transactions on Acoustics, Speech, and Signal Processing,* Vol. 37, No. 10, pp. 1550-1555, October 1989.