Object Recognition Tools for Educational Robots

Xinghao Pan
Advised by Professor David S. Touretzky

Senior Research Thesis
School of Computer Science
Carnegie Mellon University
May 2, 2008

Abstract

SIFT (Scale Invariant Feature Transform) [1] features, developed by David G. Lowe, have been found to be highly robust to translations, rotations and scaling, and have been the solution of choice for many when dealing with problems of robotic vision and object recognition. Object recognition using SIFT features involves detection of SIFT features in an image, and matching those features against features in a structured database of object images. However, to the best of our knowledge, there are currently no open-source tools to conveniently perform these tasks.

This research aims to develop SIFT-based object recognition tools for use by students in undergraduate robotics courses. The tools will allow students to gain a basic understanding of SIFT, but also abstract away from the actual implementation. Hence, students will be able to focus on solving higher level robotics problems.

Contents

1. Introduction

Motivation

The field of robotics vision and object recognition has been well studied. In recent years, SIFT (Scale Invariant Feature Transform) [1] has emerged as the solution of choice for many when dealing with problems in this area [5]. Features detected by the SIFT algorithm have been found to be invariant to transformations like scaling, rotation and translation. Object recognition involves the detection of SIFT features in an image, and the matching of those features against features in a structured database of object images.

However, despite the popularity of SIFT, there remains, to the best of our knowledge, no open source tools to conveniently perform the tasks of matching SIFT features, and constructing and maintaining the object image library. Hence, using SIFT to solve object recognition problems becomes feasible only to those who have sufficient time and resources to invest in implementing it. For students in undergraduate robotics courses, the lack of such tools becomes a major road block to solving higher level robotics problems, e.g. creating robots to play card games, that involve robotic vision and object recognition.

Goals

This research aims to develop SIFT-based object recognition tools for use by students in undergraduate robotics courses. These tools should accomplish the following aims. Firstly, the tools must allow students to not have to divert their attention to implementing SIFT, but instead focus on the problem at hand and use SIFT as a tool for solving their problems.

Secondly, as it is impossible to completely anticipate the projects that students will undertake in their courses, the tools must be generic enough to accommodate different applications.

Finally, through the use of our tools, students should be able to gain a basic understanding of how the SIFT algorithms work. This is important as different applications require different tailor-made solutions. Without understanding the strengths and shortcomings of SIFT, and also how to make adjustments within the object recognition tool, students will not be able to accomplish their goals.

Method

These aims will be accomplished by developing a C++ library of SIFT algorithms, and two primary ways for the user to interact with the library. The SIFT library will provide the functionalities of detecting SIFT features, constructing and maintaining the object library, and matching features against the object library. A Java GUI tool will allow the user to visualize SIFT features in images and to understand the matching algorithm. The user will also be able to interact with the SIFT library through an API in order to construct his own object library, and adjust the parameters according to his needs.

2. SIFT

Detecting features in an image

The SIFT feature detection algorithm consists of 4 major stages: scale-space extrema detection, feature localization, orientation assignment and generating the feature descriptor [1]. In the first 2 stages, the location of features is detected by first computing Gaussian convolutions of the image at multiple scale levels. Gaussian convolutions at adjacent scale levels are subtracted from each other to create a difference-of-Gaussians. The extremas (i.e. pixels that are minimas or maximas relative to their neighbors in the difference-of-Gaussian at the same level, and at adjacent levels) are marked as feature locations.

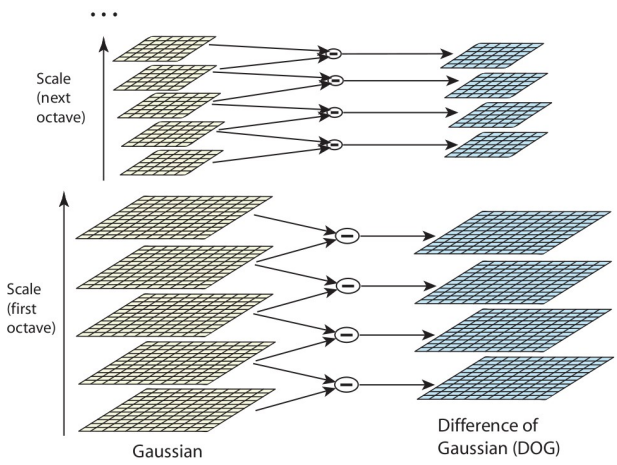

Figure 1: In first stage, difference-of-Gaussians are computed.[1]

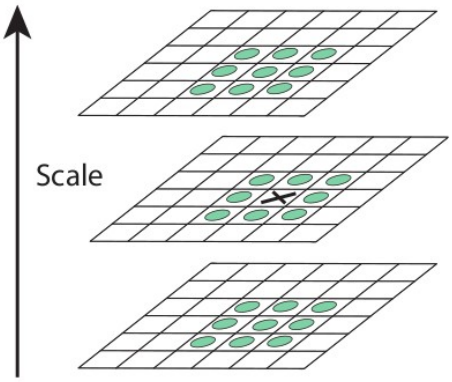

Figure 2: features are detected as extremas in DOG. [1]

Each pixel in the locality of the feature is then assigned an orientation based on its gradient in the corresponding Gaussian convolution. These orientations are weighted by a Gaussian circular window centered at the feature location and added to a histogram of orientations. Peaks in the histogram are assigned as the feature's orientations.

Finally, pixels near the feature location are divided into sub-regions. A histogram of pixels' orientations is computed for each sub-region. The feature descriptor is derived from the magnitudes of the bins of the histograms.

Since it is not an aim of this research to modify the SIFT algorithm, or to develop a feature detection technique, the full details of SIFT feature detection are left out of this paper. The interested reader may refer to [1].

Object library

The object library is organized as a hierarchical tree of objects, models and features. Thus, an object may consist of multiple models corresponding to different views of the object, and a model is

comprised of many features; on the other hand, each model belongs to a unique object, and each feature belongs to a unique model. In addition to the hierarchical tree, a k-d tree of features is maintained for ease matching features against the object library.

An object library can be iteratively constructed by training on input images. If no matches were found for the input image, a new object will be created, and the detected features will be added to the initial model of the new object. If a match was found, but the error between the input image and the model it was matched against exceeds a threshold, then a new model is created for the object of the matched model, and detected features will be added to the new model. In the third scenario, if a match was found and the error does not exceed the threshold, then the detected features are added to the matched model.

Matching features against library

The algorithm for matching images using SIFT features is described by Lowe in [2].

Individual features are first matched against a features database, which is structured as a k-d tree. Each match offers a recommendation for a possible transformation of the input image into matched image. A Hough transform hash table is used for voting for the most likely transformation. Geometric verification is performed using similarity transformation. Finally, the transformation is accepted or rejected based on an analysis of the probability of match.

3. Implementation

Our SIFT package consists of three main components: the SIFT algorithms library (which itself contains a detection module and a object library module), a Java GUI tool and the API interface. The package was implemented as part of the Tekkotsu project [3]. Tekkotsu is an application development framework for intelligent robots. It aims to give users a structure on which to build, handling routine tasks so that the user can focus on higher level programming. However, we would like to also point out that our SIFT package can be used independently of the Tekkotsu framework, and is currently available for use on Linux and Mac OS X.

Detection

SIFT++ [4] is an open-source C++ implementation of the SIFT feature detection algorithm. It was developed by Andrea Vedaldi at UCLA, and can be downloaded at http://vision.ucla.edu/~vedaldi/code/siftpp/siftpp.html. SIFT++ was incorporated into our SIFT package as the module for doing detection of SIFT features from images.

Object library

To the best of our knowledge, there are no readily available open-source implementations that perform the tasks of matching features against an object library, and constructing and maintaining that object library. Since an important method of learning an algorithm involves understanding the implementation, we decided to write our own version of the matching algorithm, thereby allowing students to explore the code in addition to the tool.

Java GUI

To facilitate the understanding of SIFT algorithms, we have created a Java GUI tool to aid visualization. This tool allows the user to create an object library by providing the program with training images. It also provides the user with information regarding detected features, matching of the image against the object library, and the organization of the object library, hence providing the user with a means of visualizing the SIFT algorithms.

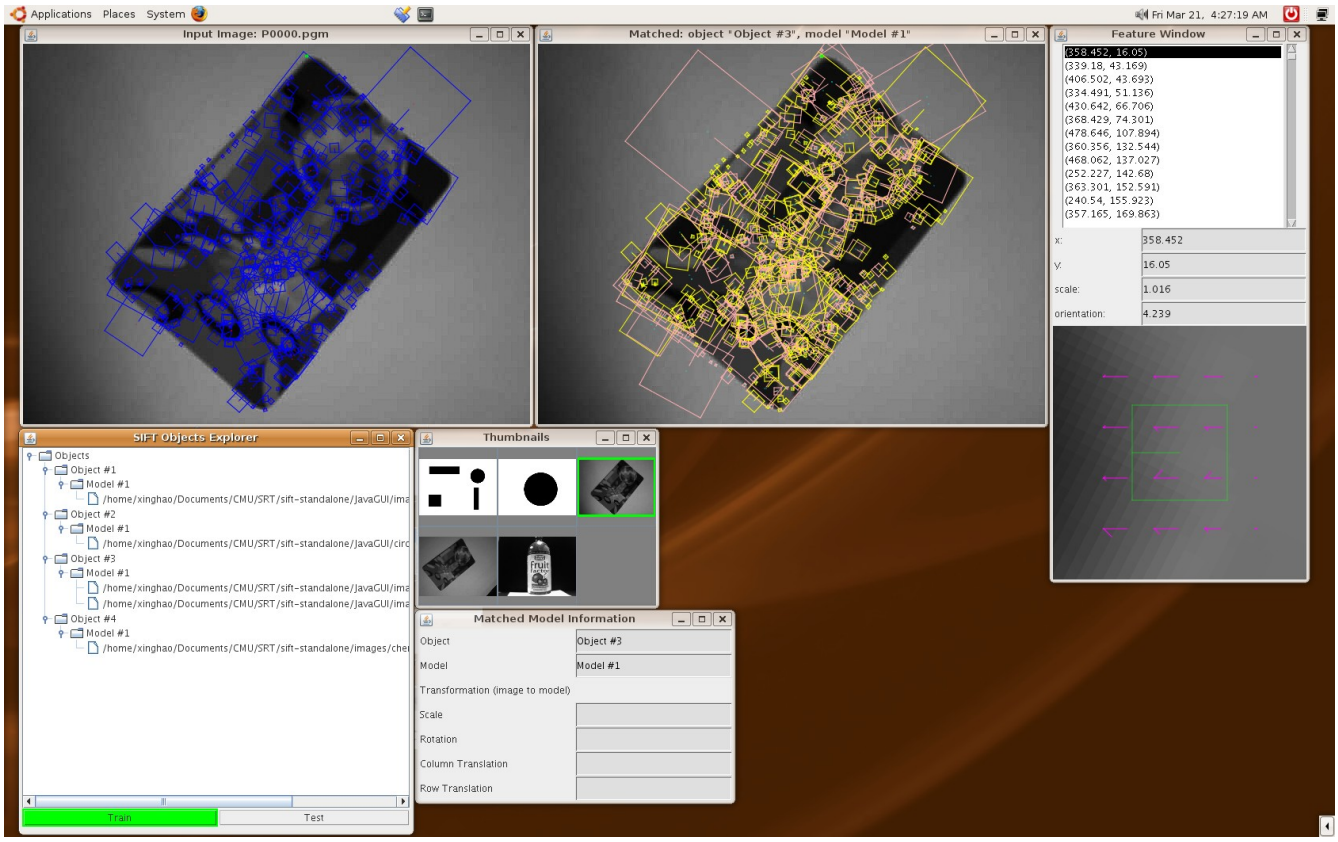

Figure 3: Java GUI tool running in Ubuntu

Each of the windows of the tool provides the user with information regarding both feature detection and matching against the object library.

Input Image Viewer

When an image is provided to the program as input, the Java GUI interacts with SIFT++ to extract features in the image. Subsequently, the features are overlaid on the displayed input image:

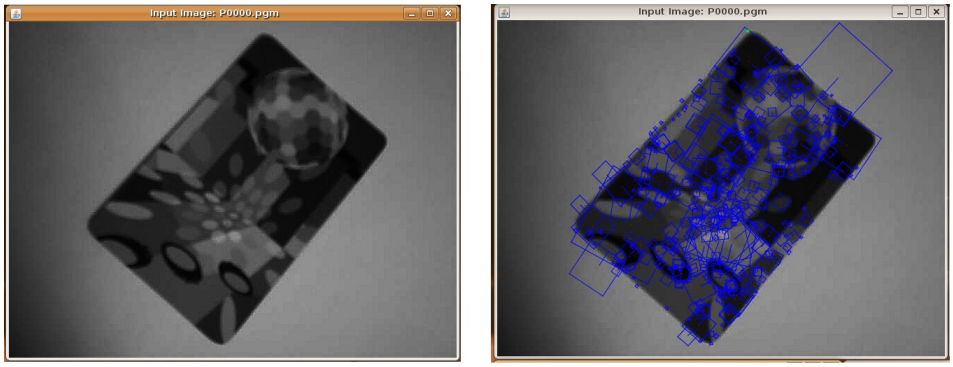

Figure 4: Input image displayed by GUI tool. The window on the left shows the original input image. On the right, SIFT features are overlaid on the image.

In the above figure, each SIFT feature is represented by a box centered on the feature's location on the input image. The size and orientation of the box corresponds to the scale and orientation of the feature respectively.

Feature Window

When the feature is selected by clicking on its location, additional information is given in the Feature Window :

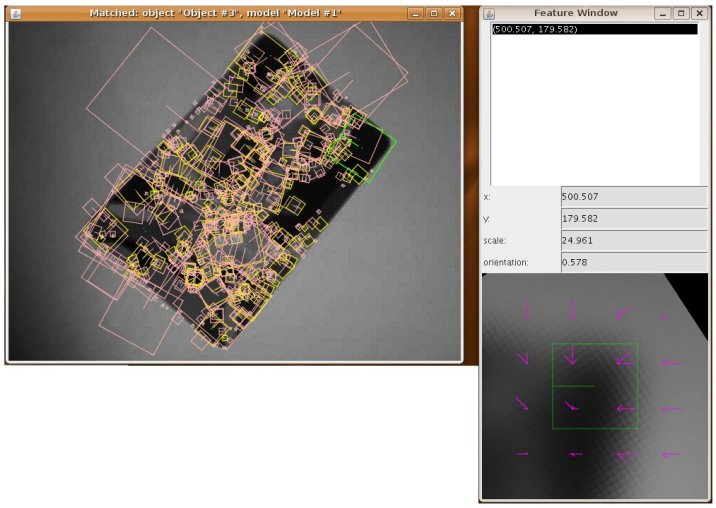

Figure 5: Feature Window shown on the right. The selected feature is in the top right corner of the card in the image. Notice that the image in the Feature Window is a Gaussian-blurred sub-image of the original, rotated relative to the feature orientation.

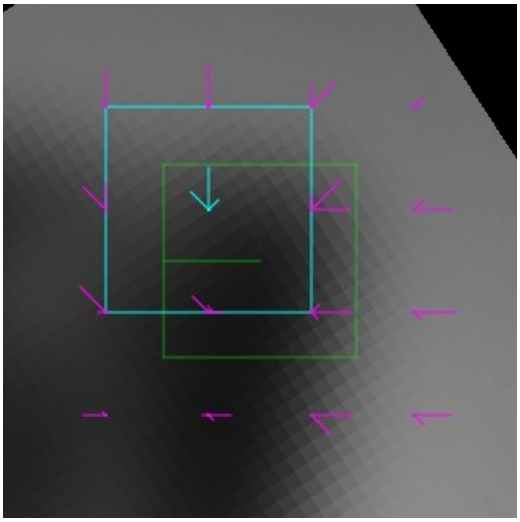

Figure 6: Gaussian-blurred image in Feature Window. The mouse pointer (not shown) is hovering over the histogram shown in cyan.

The user is able to learn of the exact location, scale and orientation of the selected feature from the

Feature Window. In a tool for learning about the SIFT algorithms, it is important to provide a means for the user to visualize the feature descriptor, and the Gaussian-blurred image from which the descriptor was computed.

The region near the feature location is Gaussian-blurred (at the of the scale of the feature), rotated relative to the feature's orientation, and displayed at the bottom of the Feature Window. The green box overlaid on the Gaussian-blurred image corresponds to the selected feature in the input image (also highlighted as a green box), thus allowing the user to visually relate the two images. 16 histograms are placed within different sub-regions of the Gaussian-blurred image, corresponding to the 16 orientation histograms that the feature descriptor is comprised of. When the user's mouse is placed over an orientation histogram, a bounding box is drawn to define the region that was used to compute the orientation histogram.

Matched Model Viewer

Our Java GUI tool offers two modes of operation. In training mode, input images are matched against models in the object image library and subsequently added to the library.

In testing mode, the input image is also matched against the object library. However, unlike in training mode, no changes are made to the object library.

In both cases, if a match is found, the GUI tool displays the matched model in the Matched Model Viewer. Features of the model are overlaid on the first image from which the model was constructed. Features of the input image that have a matching feature in the model are highlighted in the input image viewer, while their corresponding matches in the model are also highlighted. The user can select a matched feature by clicking on the feature's location, which will cause both the feature in the input image and its corresponding match in the matched model to be highlighted in green. This capability of the GUI tool facilitates the understanding of the matching algorithm.

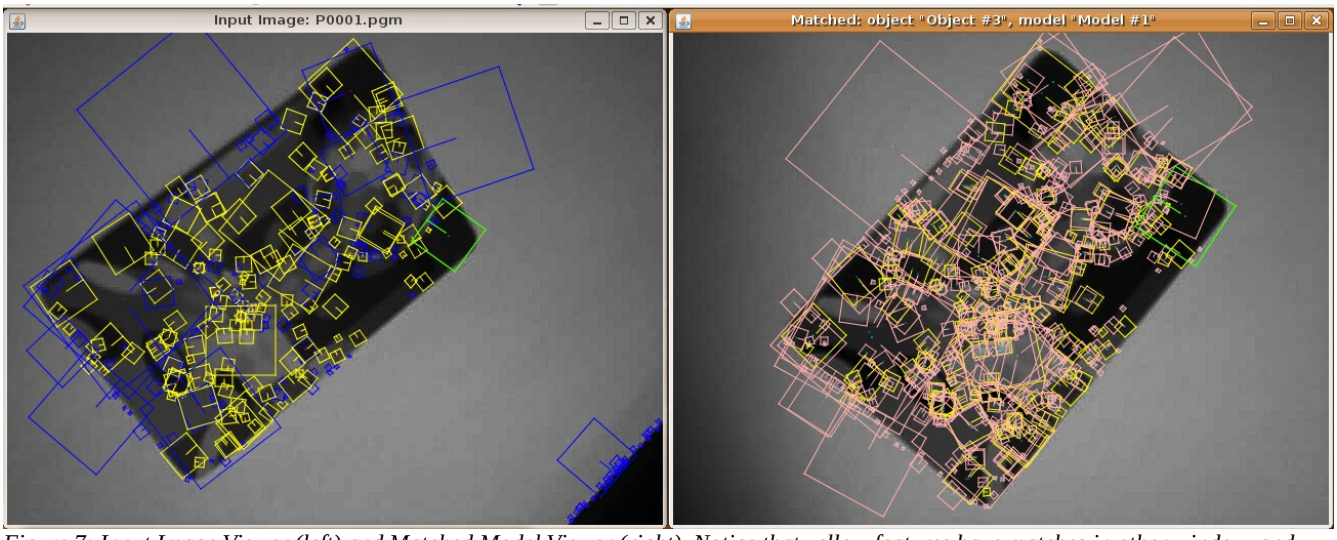

Figure 7: Input Image Viewer (left) and Matched Model Viewer (right). Notice that yellow features have matches in other window, and that the green feature in the Matched Model Viewer is the match for the green feature in the Input Image Viewer.

Matched Model Information Window

Detailed information of the similarity transformation of input image to its matching model is displayed in this window.

SIFT Objects Explorer

The object library can be constructed by iteratively training on input images. As the object library grows, the structure of the library is presented to the user as a hierarchy of objects, models and training images. Thus, the user is able to refer to this visual representation as he builds up his object library.

Thumbnail Viewer

This window displays the thumbnails of all the input images that have been used to construct the object library.

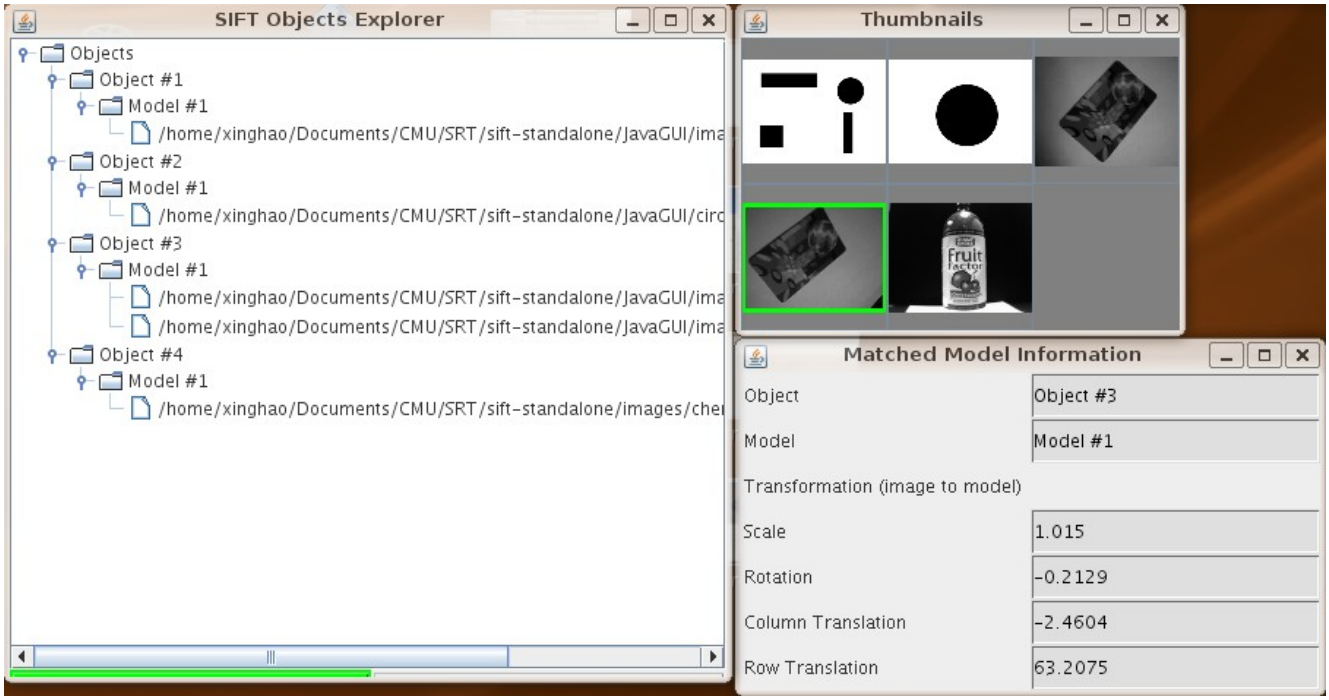

Figure 8: The SIFT Objects Explorer window is on the left, the Thumbnails viewer on the top right and the Matched Model Information window at the bottom right.

API

The API is the interface through which programmers can access the SIFT algorithms in our package within their own programs. The main C++ class that provides the API functionality is `SiftTekkotsu`, which is designed to correspond to an object library. Users of the API can build up their own library by providing it with training images, and queries can also be made about previously unseen images.

Training functions

There are three ways that training images can be provided to a `SiftTekkotsu` object library:

```
int train_addNewObject(ImageBuffer buffer);
```

> Instructs the `SiftTekkotsu` object library to incorporate the image `buffer` as a new, previously unseen object in the object library.

```
int train_addToObject(int objectID, ImageBuffer buffer);
```

> Instructs the `SiftTekkotsu` object library that the image `buffer` is an image of the object specified by `objectID`, and to incorporate `buffer` into the object library as such. The `SiftTekkotsu` object library will be responsible for determining if a new model should be created for the image or if it can be added into a known model of the specified object.

```
int train_addNewModel(int objectID, ImageBuffer buffer);
```

> Instructs the `SiftTekkotsu` object library that the image `buffer` is an image of the object specified by `objectID`, and to incorporate `buffer` into the object library as such. Use of this API call explicitly instructs `SiftTekkotsu` to create a new model for the training image `buffer`, regardless of whether there are any known models of the specified object that are similar to the image `buffer`.

Testing functions

There are two ways to query the `SiftTekkotsu` object library about training images:

```
void findObjectInImage(int objectID, ImageBuffer buffer,
vector<SiftMatch*>& matchesFound);
```

> This query is made when the user is only interested in finding a particular object, specified by `objectID`, within the test image `buffer`. All possible matches are returned as `matchesFound`, which provides information regarding the match and the transformation of image to the matched model.

```
void findAllObjectsInImage(ImageBuffer buffer, vector<SiftMatch*>&
matchesFound);
```

This query is made when the user is interested in finding all known objects in the test image `buffer`. All possible matches are returned as `matchesFound`, which provides information regarding the match and the transformation of image to the matched object model.

Miscellaneous functions

Miscellaneous functions include API calls that allow for adjustment of parameters, and saving and loading a `SiftTekkotsu` object library to and from a file.

```
void   setParameter(const char* paramName, double paramVal);
double getParameter(const char* paramName);
```

This pair of functions provide a means of tuning parameters according to the user's needs. Some of the currently supported parameters include `probOfMatch`, which determines the minimum probability of accepting a possible match of an object to the image, and `errorThreshold`, which determines the maximum acceptable error of the match between the image and the object before the match is rejected.
There is a need for users to be able to control these parameters both to aid their understanding of the effects of the parameters on SIFT, and to tailor-make a solution to their unique problems.

```
void saveToFile(const char* filename);
void loadFile(  const char* filename);
```

The `SiftTekkotsu` object library can be exported and imported from files by using this pair of API calls. Training of a substantially large `SiftTekkotsu` object library takes a considerable amount of time. Being able to import and export the `SiftTekkotsu` object library allows the user to perform a single training run, and to save the `SiftTekkotsu` object library for future object recognition tasks.

4. Evaluation

Workshops

The Java GUI tool was presented at a recently concluded SIGCSE 2008 Workshop on Computer Vision and Image Processing: Accessible Resources for Undergraduate Computer Science and at a separate workshop conducted by Prof. David S. Touretzky at Spelman College. Prof. Tourtezky reported that it was well received by the participants of the workshops.

Cognitive Robotics course

Our SIFT package was released for use in the Cognitive Robotics course that is being conducted at Carnegie Mellon University in the Spring 2008 semester. As part of the course requirements, students are expected to complete a class project. One of the students chose to do a project that involved the use of the SIFT package, and we're hoping to get feedback from her by the end of the semester.

During the course of the project, we encountered an unanticipated setback in that the object libraries constructed by the SIFT tools were too large to be placed on the AIBO robot's memory stick. Thus, AIBO programmers who wish to use the tool must rely on off-board processing, which is supported by Tekkotsu. However, more modern robots that don't rely on memory sticks are not subjected to this limitation.

5. Future work

The evaluation of the SIFT package within the classroom environment is still an ongoing project. The SIFT package will continue to be made available to students in undergraduate robotics courses, and further improvements will be made to the package based on the feedback we receive from both the students and the instructors of these courses.

Another area of research that we are considering is to combine color-based vision techniques with our SIFT package. There are tools in the Tekkotsu framework that can be used for color segmentation. These tools may be utilized for implementing color-based vision techniques. There is also a possibility of modifying SIFT itself into a color-based algorithm instead of one based on orientations.

6. References:

1. Lowe, D. G., (2004) Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*
2. Lowe, D. G. (2001) Local Feature View Clustering for 3D Object Recognition. *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*
3. Tekkotsu Homepage: http://www.tekkotsu.org/
4. Vadaldi, A., SIFT++, http://vision.ucla.edu/~vedaldi/code/siftpp/siftpp.html
5. Scale-invariant feature transform, *Wikipedia*, http://en.wikipedia.org/wiki/Scale-invariant_feature_transform