**Object Recognition Tools for Educational Robots**

Xinghao Pan
Advised by Professor David S. Touretzky

**Abstract**

SIFT features are the solution of choice for many when dealing with problems of robotic vision and object recognition. However, to the best of our knowledge, there are no open source tools to conveniently perform the tasks of matching features, and constructing and maintaining the image library. This research aims to develop SIFT-based robotic object recognition tools for students in undergraduate robotic courses. The tool will be evaluated within a real classroom setting when released for use in a Carnegie Mellon Cognitive Robotics course.

## 1. Introduction

SIFT (scale-invariant feature transform) [1] is a feature detection algorithm that was developed by David G. Lowe. Features detected by the SIFT algorithm have been found to be robust to translations, rotations and scaling. As such, SIFT features have become the solution of choice for many when dealing with problems of robotic vision and object recognition.

Object recognition using SIFT features is accomplished in two phases. Features are first detected using Lowe's SIFT algorithm, and then matched against features in a structured database of object images. While there is an open-source implementation of SIFT feature detection, to the best of our knowledge, there are no open source tools to conveniently perform the the tasks of matching SIFT features, and constructing and maintaining the object image library.

The aim of this research is to construct such tools for robotic vision and object recognition for students in undergraduate robotic courses. This would involve allowing programmers to use and also to understand the basics of the algorithms behind the tool, and to make adjustments within the object recognition tool to accomplish their goals. As part of the research, the tool will be evaluated within a real classroom setting when it is released for use in an undergraduate cognitive robotics course.

### *1.1. SIFT*

The SIFT feature detection algorithm consists of 4 major stages: scale-space extrema detection, keypoint localization, orientation assignment and generating the keypoint descriptor [1]. In the first 2 stages, the location of keypoints 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 keypoint locations.

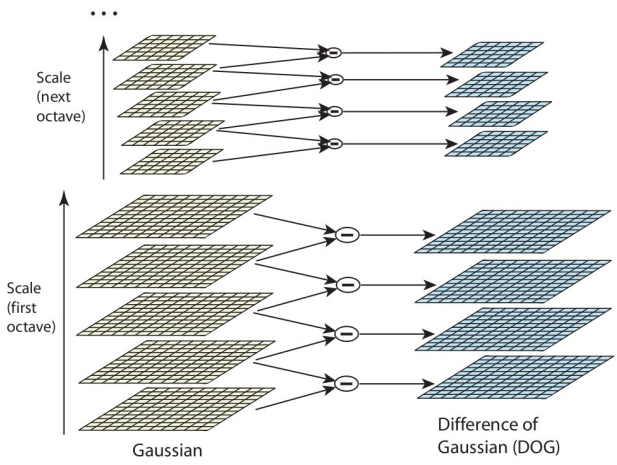


*Figure 1: In first stage, difference-of-Gaussians are computed.[1]*

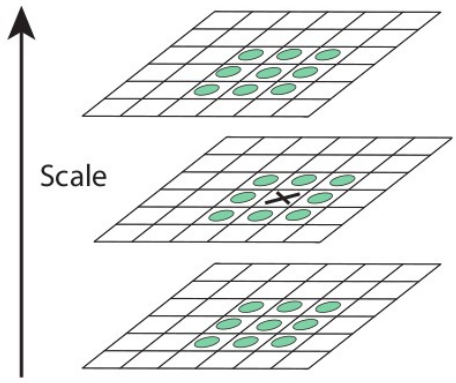


*Figure 2: Keypoints are detected as extremas in DOG. [1]*

Each pixel in the locality of the keypoint 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 keypoint location and added to a histogram of orientations. Peaks in the histogram are assigned as the keypoint's orientations.

Finally, pixels near the keypoint location are divided into sub-regions. A histogram of pixels' orientations is computed for each sub-region. The keypoint 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].

*1.2. Matching images using SIFT features*

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.

For details, the interested reader may refer to [2].

*1.3. Object Library*

An object library can be iteratively constructed by training on input images. The library is organized as a hierarchical tree of objects, models and keypoints. Thus, an object may consist of multiple models corresponding to different views of the object, and a model is comprised of many keypoints; on the other hand, each model belongs to a unique object, and each keypoint belongs to a unique model.

Upon training on a new input image, one of the following three things will occur. If no matches were found for the input image, a new object will be created, and the detected keypoints 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 keypoints 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 keypoints are added to the matched model.

*1.4. SIFT for Tekkotsu*

As previously stated, our goal is to create a tool for students in undergraduate robotics course to use and understand the SIFT algorithms. To accomplish this aim, we require implementations of SIFT feature detection and matching. While there is an open-source implementation of feature detection, there are no readily available implementations of the matching algorithm. 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.

We have also created an Java based GUI tool that allows the user to create an object library by providing the program with training images. The GUI tool 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.

All these tools have been packaged together 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.

**2. Implementation**

*2.1. SIFT++*

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.

*2.2. Java GUI Visualization Tool*

To facilitate the understanding of SIFT algorithms, we have created a Java GUI tool to aid visualization:

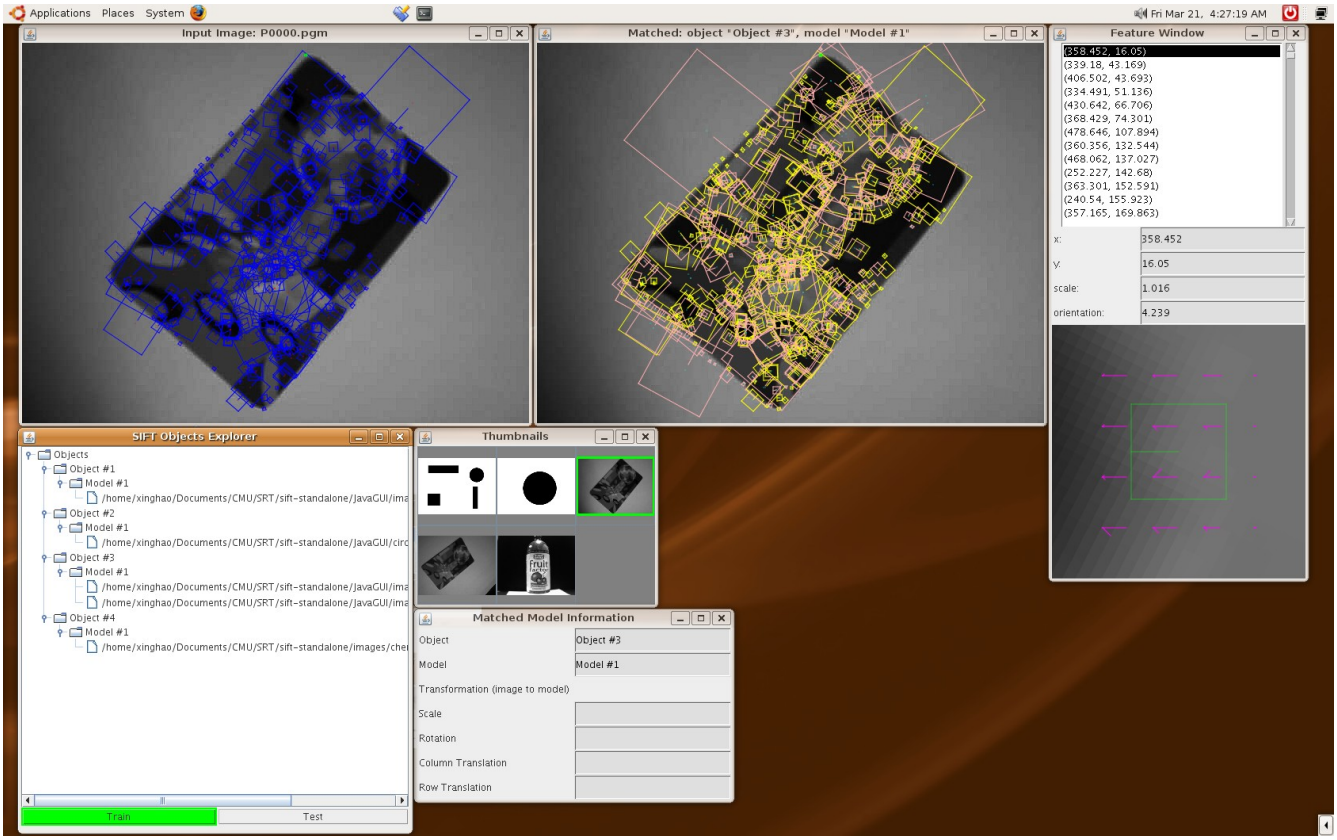

*Figure 3: Java GUI tool running in Ubuntu*

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

*2.2.1. 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:

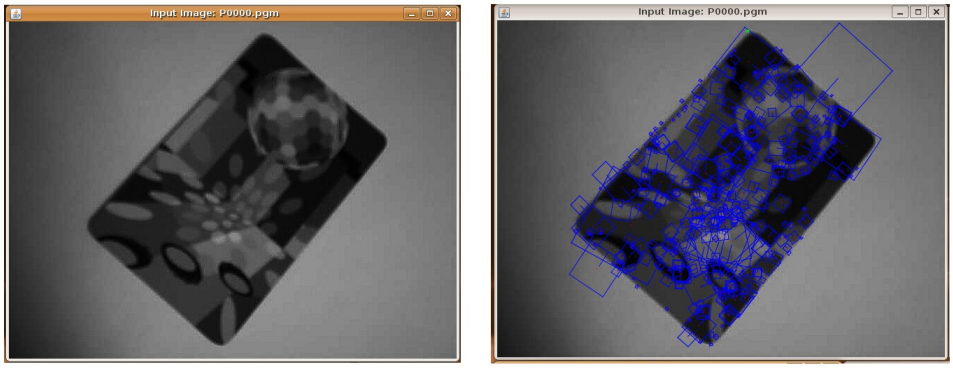

*Figure 4: Input image displayed by GUI tool. The window on the left shows the original input image. On the right, SIFT keypoints are overlaid on the image.*

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

*2.2.2. Feature Window*

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

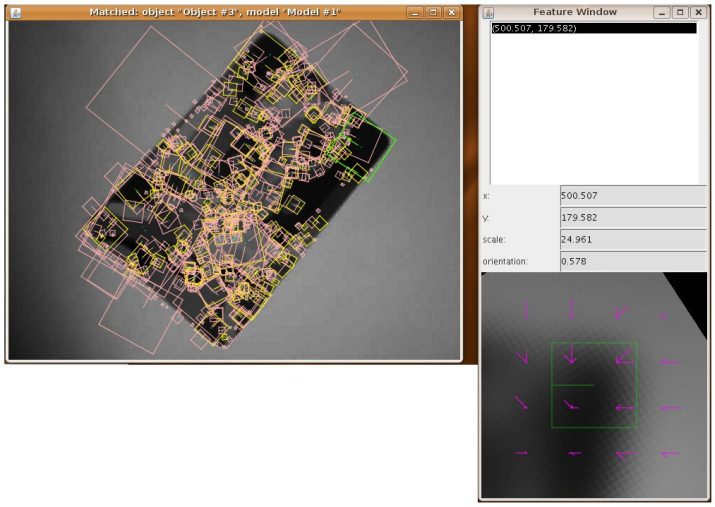


*Figure 5: Feature Window shown on the right. The selected feature is in the top right corner of the card in the image. Notice that the image in the Feature Window is a Gaussian-blurred sub-image of the original, rotated relative to the feature orientation.*

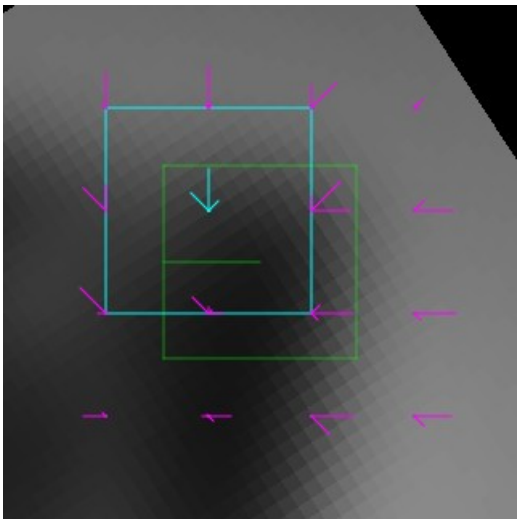

*Figure 6: Gaussian-blurred image in Feature Window. The mouse pointer (not shown) is hovering over the histogram shown in cyan.*

The user is able to learn of the exact location, scale and orientation of the selected keypoint 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 keypoint descriptor, and the Gaussian-blurred image from which the descriptor was computed.

The region near the keypoint location is Gaussian-blurred (at the of the scale of the keypoint), rotated relative to the keypoint's orientation, and displayed at the bottom of the Feature Window. The green box overlaid on the Gaussian-blurred image corresponds to the selected keypoint 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 keypoint 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.

*2.2.3. 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. Keypoints of the input image that have a matching keypoint 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 keypoint by clicking on the keypoint's location, which will cause both the keypoint 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.

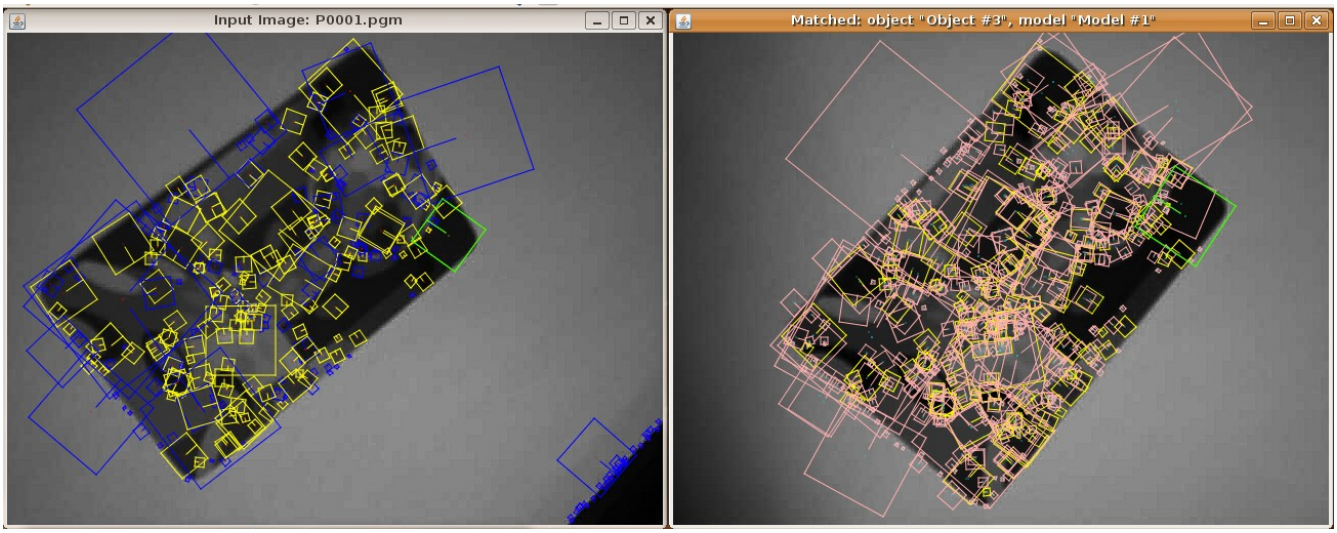


*Figure 7: Input Image Viewer (left) and Matched Model Viewer (right). Notice that yellow features have matches in other window, and that the green keypoint in the Matched Model Viewer is the match for the green keypoint in the Input Image Viewer.*

*2.2.4. Matched Model Information Window*

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

*2.2.5. 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.

*2.2.6. Thumbnail Viewer*

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

**3. Evaluation**

*3.1. SIGCSE 08*

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 . During the hands-on session, two Computer Science professors, Zach Dodds of Harvey Mudd College and Deborah Burhans of Canisius College, were able to personally try out the GUI tool. They were impressed with the performance of the object recognition, and are considering incorporating it into the classes that they are teaching.

*3.2. Cognitive Robotics Course*

We intend to release the SIFT for Tekkotsu package to students currently attending the Cognitive Robotics course at Carnegie Mellon University. The tool will be evaluated based on the students' responses and feedback.

**4. Future Work**

Currently, we are developing an API for constructing and maintaining a SIFT-based object library. This will be completed before releasing the package to students in the Cognitive Robotics course.

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.

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