Toward Real-Time Grocery Detection for the Visually Impaired

Tess Winlock, Eric Christiansen, Serge Belongie
Computer Science & Engineering
University of California, San Diego
{twinlock, echristiansen, sjb}@cs.ucsd.edu

Abstract

We present a study on grocery detection using our object detection system, ShelfScanner, which seeks to allow a visually impaired user to shop at a grocery store without additional human assistance. ShelfScanner allows online detection of items on a shopping list, in video streams in which some or all items could appear simultaneously. To deal with the scale of the object detection task, the system leverages the approximate planarity of grocery store shelves to build a mosaic in real time using an optical flow algorithm. The system is then free to use any object detection algorithm without incurring a loss of data due to processing time. For purposes of speed we use a multiclass naive-Bayes classifier inspired by NIMBLE, which is trained on enhanced SURF descriptors extracted from images in the GroZi-120 dataset. It is then used to compute per-class probability distributions on video keypoints for final classification. Our results suggest ShelfScanner could be useful in cases where high-quality training data is available.

1. Introduction

The problem of specific object detection in video streams is well established in computer vision literature. For certain objects, various classifiers including boosted Haar-like wavelets [13], SIFT keypoint matching [7], and random forests for classification [12] all show very good performance on this task. We focus on real-time product detection from mobile video in grocery stores, with the intent of enabling a blind user to scan a scene with a camera and have it detect items on her shopping list. This problem presents difficulties for reasons beyond the standard perspective, occlusion, and specularity distortions that exist in natural scenes. This is because product appearances have a tendency to change often and algorithms must be efficient enough to run quickly on constrained hardware.

Standard object detection algorithms require some amount of training data. The best performance is often found when the training data are sampled from the same distribution as the testing data. In our problem domain, in which object detection algorithms are used to help the visually impaired shop in a grocery store, it is infeasible to maintain equality between the training and testing distributions. This is because product packaging changes regularly, and so maintaining equality between the distributions would require constant updating of the training examples. We instead work in the paradigm explored by [9], in which training and testing data are drawn from entirely different distributions. For training data, they use **in vitro** images, which were gathered from the Web and were taken under ideal lighting and perspective conditions. For testing data, they use **in situ** images[1], which were obtained from video streams captured in a grocery store, and are thus subject to natural conditions such as occlusion, specularity, and perspective distortion. We use the same dataset, the GroZi-120, and we work in the same paradigm, using in vitro images of items on a user's shopping list to detect items in situ.

Detection of items from a shopping list is a problem previously unexplored in the study of the GroZi-120. A usable system would likely need a multiclass detector which would be capable of detecting the items on the shopping list and which would be efficient enough to process the incoming stream of video frames in real time. In the case of many object detection applications, such as pedestrian detection, this requires keypoint extraction and matching from full frames, in which each frame is processed independently of the other frames. This approach is necessary when the state of the world could change drastically from one frame to the next; a pedestrian could move or a car could veer in the way. In the case of grocery store shelves[2], the scene is both static and mostly planar; thus, we can assume minimal change in the scene from one frame to the next. This allows ShelfScanner to compute a local image registration using standard optical flow techniques, and only extract keypoints from pixels known to be new to the system. For simplicity our system

[1] In the language of the LFW database, these images would be "in the wild" [5].

[2] In this study we focus on standard product shelves; produce and deli sections are not included.

978-1-4244-7028-0/10/$26.00 ©2010 IEEE

assumes a translational motion model for local image alignment, but it can be readily extended to more general models such as affine or projective [6]. This enables ShelfScanner to process less data and run more quickly.

This paper presents an approach taken on the GroZi-120 dataset to detect items from 10-item grocery lists. The remainder of this paper is organized as follows: Section 2 details related approaches, Section 3 details ShelfScanner's approach and methods, Section 4 outlines our experimental construction, and finally Sections 5 and 6 discuss our results.

2. Related work

2.1. Object detection algorithms

A commonly used method for object detection, feature point descriptor matching relies upon extracting robust features from training data that can be accurately matched to features extracted from testing data. Among the more commonly used descriptor extraction algorithms is Lowe's SIFT, which outperforms most others with its invariance to both scale and local deformations. It does so by capturing large amounts of local spatial intensity pattern information which it then stores in the form of a 128 bin histogram. The robust nature of these features has been shown to be very useful for interest point matching in object detection applications, including previous studies of the GroZi-120 dataset [9].

Bay's SURF descriptors [3] have robust qualities reminiscent of SIFT descriptors and can be computed faster. The descriptor works very similarly to that of SIFT, in that it places local intensity information about the interest points into histograms. Its approach relies on computing the responses of many Haar-like wavelets within the neighborhood of the feature point, which are faster to compute than gradients. A recommended object detection strategy, as taken by Bay, is to match keypoints found in the test image to keypoints in the bag of training keypoints and then fit a homography to the matched points to determine an object's location in an image. This approach works well for images with well registered keypoints and a single item search. However, we have found that computing a homography through standard RANSAC [4] is too computationally expensive to be applied to every item in a shopping list in parallel, and is error prone due to the approximate nature of our mosaicing, which we discuss further in Section 3.3.

A computationally cheaper method that lends itself to descriptor-based object recognition is the naive Bayes model used by NIMBLE [2]. In NIMBLE, an image is assumed to contain exactly one item from exactly one class, and the descriptors extracted from the image are assumed to be conditionally independent of each other given the class of the image. NIMBLE uses nonparametric density estimation techniques to estimate the probability of a descriptor given a class. Through Bayes' theorem and the conditional independence assumption, one can estimate the probability that an image corresponds to a particular class given all the descriptors extracted from the image.

Previous work on this dataset provides a performance baseline with results for SIFT, Viola and Jones, and color histogram matching [9]. They showed that on average SIFT performed the best, due to its invariance to scale, rotation and other factors. Color histogram matching showed very good performance, which is justified as grocery store products tend to be colored to attract attention and distinguish themselves from competitors. The Viola and Jones method had a performance only slightly better than random.

2.2. Mosaicing

Building mosaics from video sequences has been explored for many years in various applications. Irani [6] described two general approaches for mosaic frame registration: static and dynamic. Static refers to offline global frame registration, and dynamic refers to online frame registration. A 2D alignment scheme is used which places each image in the mosaic relative to either the first frame or a mosaic coordinate system. The choice of motion models for the camera can range significantly depending on the application. Simple affine models can be used for translational motion cases, or more general perspective models can be used to account for significant parallax and perspective distortion. The final step in Irani's method is to improve the mosaic by performing image blending.

A faster approach as described by Adams, Gelfand, and Pulli [1], is to align subsequent images using integral projections of edges. This approach extracts these integrals and then computes the least squares estimate of the best similarity transform (rotation, uniform scale, and translation). They have shown that this can run at 30 frames per second (FPS) on a standard smart phone, which shows promise in moving ShelfScanner onto this platform.

3. Our approach

ShelfScanner assumes the user holds a camera and sweeps the camera's field of view (FOV) across grocery shelves. This restriction stems from our translational model, which will be aleviated in future work when we move to a more general motion model. When the FOV contains an object on the user's shopping list, ShelfScanner should detect the object, with the purpose of notifying the user.

The total input to the system is:

1. Images of items from the user-supplied shopping list. These images come from the in vitro subset of the GroZi-120 dataset. The in vitro images are taken from the Web, and the images usually show ideal specimens

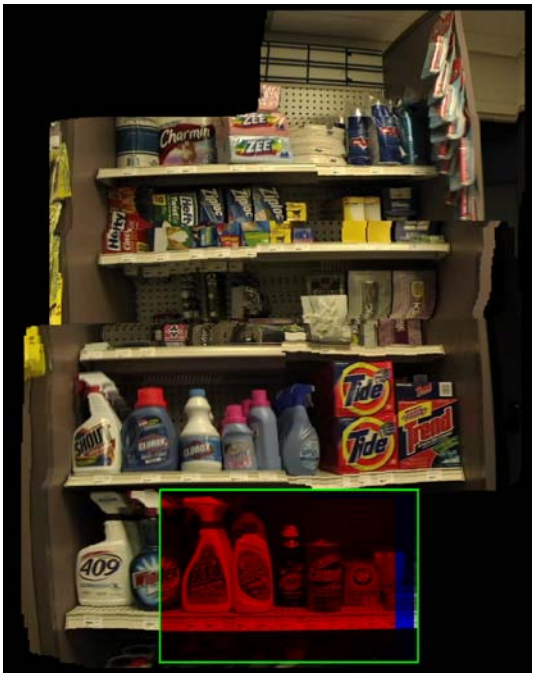

Figure 1: A mosaic under construction. The green box is the current FOV. The black space represents parts of our scene we have not yet explored. The blue-tinted pixels are pixels that were revealed for the first time by this FOV. The red-tinted pixels lie in a region we have already seen, but will be updated by this FOV.

of each product, taken under ideal conditions. The GroZi-120 dataset supplies between 2 and 14 in vitro images per item, with an average of 5.6 images per item.

2. A video stream taken from the user's camera as she sweeps its FOV along shelves. The GroZi-120 video has resolution 720x480.

The total output of the system is a set of points from frames in the video stream that ShelfScanner thinks are likely to belong to items in the shopping list. ShelfScanner finds these points in an online manner, so the user could theoretically be notified that the camera FOV contained a shopping list item mere moments after sweeping the FOV across the item. See 5.2 for more information on real-time performance. ShelfScanner also maintains a sense of where the current FOV is with respect to previous FOVs by building a mosaic as it runs; see Figure 1 for an example. This global position information could be used to direct the user back to an item if the item has already passed out of the current FOV.

ShelfScanner consists of an offline training component and an online evaluation component. Each cycle of the evaluation component is driven by the arrival of a new frame from the camera, and the actions performed in this cycle are as follows:

1. **Mosaicing** 3.2. The new frame is added to the mosaic. Besides extending the mosaic to include new data, this allows us to determine which regions of the new frame have been seen before, and can therefore be ignored, and which regions are new, and must therefore be examined.

2. **Interest point detection and descriptor extraction** 3.3. We use SURF to identify interest points in the unseen regions of the current frame, and we extract a ShelfScanner descriptor from each interest point. The point from which the descriptor was extracted, along with the descriptor itself, will be called a **ShelfScanner keypoint**.

3. **Descriptor-specific probability distributions** 3.4. For each new keypoint, we estimate a distribution over shopping list classes *given the keypoint*.

4. **Window probability distributions** 3.5. For each new keypoint, we use the distributions estimated in the previous step to estimate a distribution over shopping list classes *given all the keypoints within a radius of this keypoint*.

5. **Keypoint selection** 3.6. Of the keypoints whose window probability distributions were just calculated, we threshold by estimated class probabilities to select a subset, where each keypoint in this subset is believed to correspond to an item on our shopping list. These keypoints give locations on the shelf where the user should search for items from the shopping list.

3.1. Training

Our training set consists of all the images of items from the shopping list, plus some fixed **background** images. The background images are photos from the inside of a different grocery store that don't contain any possible shopping list items. In our setup, the background images are thought to be training examples for their own class, the background class. Each training image is processed with SURF, which provides us with a set of descriptors for each image. We use a Hessian threshold of 500 and a window size of radius 20 pixels. In addition, we calculate a color histogram at the location of each descriptor, and append it to the end of the descriptor. For brevity, we will call the concatenation of the SURF descriptor and color histogram a **ShelfScanner descriptor**. Because luminance varies considerably between the in vitro training images and the footage from the store, we use the L*a*b* color representation scheme, and discard the lightness "L". Using a window width of 21, we compute histograms containing 16 bins for the remaining *a*

and b channels, yielding a total of 32 additional dimensions for the descriptor. To fight redundancy and for later computational expediency, we reduce the number of descriptor dimensions from $64 + 32 = 96$ to 25 using PCA. The principal components are calculated using the descriptors from all the images.

We then combine descriptors from images of the same class, producing a bag of descriptors for each class. These bags of descriptors will later be used for nearest-neighbor based nonparametric density estimation, so for each class we initialize the randomized k$_d$-tree algorithm from its bag of descriptors, using OpenCV's interface to the FLANN library [10].

3.2. Mosaicing

The problem domain of a grocery store shelf is suited to building mosaics using the Lucas-Kanade optical flow method [8], as implemented by OpenCV[3]. Unlike many other natural scenes, grocery store shelves are generally static; the products and shelves seldom move during a shopper's time spent in the store. In addition, products generally sit on the edges of shelves to make them as visible as possible. This forms a fairly planar, static surface. When the camera's FOV translates along the shelves, as is the case in 22 of the 29 videos in the GroZi-120 dataset, we can use optical flow and a simple translational model to paint a mosaic, as in Figure 1.

Given a new frame, we extract features using the approach described by Shi and Tomasi [11], and then use optical flow to determine each feature point's relative translation from the last FOV. We found that an individual estimated translation between feature points is a very noisy estimate of the true translation between the FOVs, but by taking the median[4] of all the individual translations we can get a fairly accurate estimate of the true translation, as assessed qualitatively by the authors. Given the translation with respect to the last frame, we can determine the location of the current FOV in the mosaic. Finally, we overwrite the mosaic in these coordinates with the pixel values from the current FOV.

3.3. Interest point detection and descriptor extraction

In the previous step we calculated the mosaic coordinates of the current FOV. Using this knowledge, and knowing which pixels in the mosaic had been written to *before* the last frame was added, we can determine which regions in the current FOV are new, and which were viewed in a previous frame. Since regions that were viewed in previous frames have already been searched for interest points, we don't need to search again. Instead, we can confine our attention to new regions of the scene. Thus, we extract SURF descriptors and color histograms (together, ShelfScanner descriptors) only from previously unseen regions in the FOV. Because ShelfScanner descriptors rely on windows of width 41 about a keypoint, we must ensure we do not extract a descriptor if its keypoint is within 20 pixels of the current border of the mosaic. We maintain two binary masks with the same dimensions as the mosaic to ensure that we only process new pixels. The first mask, or new data mask, has a value of one at the location of each new pixel that we consider for extracting a SURF descriptor. The second mask, or processed data mask, has a value of one at the location of each pixel we have previously considered. Once a new frame's location in the mosaic is known, the new data mask is computed for every pixel in the new frame that is not contained in the processed mask and is at least a distance of 20 pixels away from the edge of the frame. This ensures that keypoints will not be extracted from the edge of the image. After SURF keypoints have been extracted, all pixels have been considered so the new data mask is added to the processed mask.

3.4. Descriptor-specific probability distributions

We reduce the dimension of each new keypoint by projecting it onto the PCA vectors found in the training phase. For each projected keypoint, we estimate its density under the distributions for each of the items on our shopping list, as well as its density under the background class. Following an approach taken by NIMBLE [2], we obtain the density estimates using the k-nearest-neighbor kernel, with $k = 1$:

$$p(\mathbf{d}|c) \propto \frac{1}{T_c \cdot V}. \tag{1}$$

Here, $\mathbf{d}$ is the projected ShelfScanner descriptor, c is a class of items, T_c is the number of training examples in class c, and V is the volume of the smallest sphere centered at $\mathbf{d}$ that contains a training example in class c. Note $p(\mathbf{d}|c)$ can be thought of as the probability of extracting the descriptor $\mathbf{d}$ if we randomly extract a descriptor from an image of class c.

We assume a uniform prior over classes, and so get

$$p(c|\mathbf{d}) \propto p(\mathbf{d}|c). \tag{2}$$

So by normalizing the values obtained in the previous step, we obtain an estimate for each descriptor $\mathbf{d}$ that its keypoint overlays an item of a particular class; this is the **descriptor-specific probability distribution**. In Figure 2, we display these estimates for the Tide and the Arm & Hammer classes in the form of a **heatmap**, where each keypoint is plotted as a colored pixel, the color of which reflects the amount of weight placed on the given class by the

[3] http://opencv.willowgarage.com/wiki/

[4] Note the median of a set of n dimensional vectors is constructed by taking n component-wise medians.

keypoint's descriptor-specific probability distribution. Note these heatmaps are fairly noisy; not all the bright points are clustered around the images to which they correspond. This inspires the next step, in which noise will be reduced using windows.

3.5. Window probability distributions

As mentioned above, the descriptor-specific probability distributions that we obtained in the previous step are noisy. In particular, the distributions have high spatial variance; two nearby keypoints are likely to correspond to the same class, but may easily have very different descriptor-specific probability distributions, which is due in part to the high-variance nature of nonparametric density estimation. We create a spatially smoother and more accurate class distribution for each keypoint by taking advantage of keypoint proximity information.

In particular, we consider a window of width 41 centered at each keypoint, and employ the approach taken by NIMBLE. We assume, usually falsely, that the window contains exactly one class from our shopping list, and we calculate the probability of each of the classes given the keypoints in the window, using flat priors and a naive Bayes assumption. If D is the set of descriptors of the keypoints in the window, C is the set of classes in the shopping list, including the background, and $c \in C$, we have

$$p(c|D) = \frac{p(D|c)}{\sum_{c' \in C} p(D|c')} = \frac{\prod_{\mathbf{d} \in D} p(\mathbf{d}|c)}{\sum_{c' \in C} \prod_{\mathbf{d} \in D} p(\mathbf{d}|c')}. \quad (3)$$

Here the second equality follows from the naive Bayes assumption.

Note that in the previous step, we computed all the quantities in the right-most expression of Equation 3. Thus for each class c, we can calculate $p(c|D)$ and thus obtain a **window probability distribution**. This window probability distribution is analogous to the descriptor-specific probability distribution, but it is usually much more accurate. Heatmaps computed with window probability distributions can be found in Figure 3. Note the qualitative improvement over the heatmaps from Figure 2, which were computed with descriptor-specific probability distributions.

Note that we can compute a window probability distribution for a keypoint only if we have descriptor-specific probabilities for all its neighbors within a 20 pixel radius. This is analogous to the situation described in Section 3.3, and so we use an analogous method.

3.6. Keypoint selection

At this stage, we decide which keypoints the user should investigate. We consider the set of keypoints with newly estimated window probability distributions, and for each keypoint and each shopping list class, we look at the amount

(a) Arm & Hammer heatmap

(b) Tide heatmap

Figure 2: Heatmaps generated for the Arm & Hammer and Tide classes with *descriptor-specific probability distributions*. The redder the keypoint, the more its descriptor-specific distribution believes that it overlays an item of the given class. If you find it difficult to see the underlying items, refer to Figure 1, which shows the same scene. Note these heatmaps are noisier than those given in Figure 3.

(a) Arm & Hammer heatmap

(b) Tide heatmap

Figure 3: Heatmaps generated for the Arm & Hammer and Tide classes with *window probability distributions*. The redder the keypoint, the more its window distribution believes that it overlays an item of the given class. If you find it difficult to see the underlying items, refer to Figure 1, which is of the same scene. Note these heatmaps are cleaner than those given in Figure 2.

of weight the keypoint's window distribution places on the class. If the weight is above some threshold, we recommend that the user search for the item at the keypoint. This threshold is a parameter of ShelfScanner.

In addition, to prevent ShelfScanner from recommending keypoints very near each other, as they are likely to come from the same item, we remove from future consideration any keypoints that lay within 20 pixels of a previously recommended keypoint.

4. Experiments

ShelfScanner is intended to run in real-time, so we conducted experiments to measure computational performance as well as the accuracy of the recommendations.

4.1. Recommendation performance

For a particular class, we first ask whether ShelfScanner will ever recommend a keypoint that overlays an item of that class. Second, we ask how many keypoints are mistakenly recommended before a correct keypoint is recommended. If we assume the user can navigate to the item whose keypoint we recommend, and has some way of verifying the identity of an item, such as with a hand-held barcode scanner, then the first question asks whether the user would ever find the item, and the second asks how many items she would investigate before finding her item. This is closely related to precision and recall, but more focused for our application. With the mosaic providing a global coordinate system, we believe it would be straightforward to navigate the user to the item whose keypoint we recommend, and hand-held barcode scanner software such as RedLaser[5] for the iPhone or photo object recognition software such as LookTel[6] for Window's Mobile can be obtained for smartphones.

We choose 10 videos from the dataset that are free of cutting between shots and in which the camera approximately adheres to a translational motion model. We then randomly select 10 items for the shopping list from the 52 items that appear in these 10 videos and run ShelfScanner with this shopping list as an input. We execute this last step a total of 25 times, randomly generating a new shopping list each time.

Before we could run our experiments, we had to label the test data. We did this by labeling the regions of the generated mosaic that correspond to each item class.

4.2. Computational performance

For all ten of the videos from the recommendation performance experiment, we run ShelfScanner with a random shopping list and measure how much time it takes, on average, to process a new FOV. As the current system is meant

[5] http://redlaser.com/
[6] http://www.looktel.com/

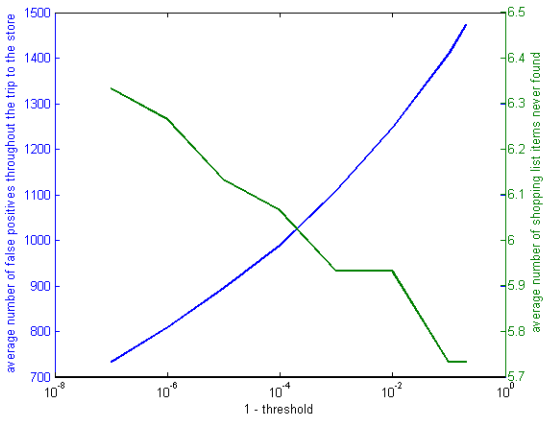

Figure 4: The right axis shows the average number of mistakenly recommended keypoints (false positives), the left shows the average number of shopping list items that are not retrieved by the user in a visit to the store. Note with a threshold of .9999, the user should expect to find around 4 of her 10 shopping list items, and investigate around 1000 items that are not on her shopping list.

to be run on a powerful laptop worn on the the user's back, the timing data comes from a single core on an Intel Core 2 processor running at 2.4GHz computer running a 64-bit ArchLinux with 8 GB of RAM.

5. Results

5.1. Recommendation performance

Plots of our two metrics as a function of threshold can be found in Figure 4. Naturally, as the threshold increases, the risk of missing items on the shopping list increases and the number of mistakenly recommended keypoints (false positives) is reduced. Unfortunately, there does not exist a threshold where a user is likely to retrieve a majority of her shopping list items and be subject to a reasonable number of false positives.

To understand the difficulty ShelfScanner has with this dataset, we investigate the differences between classes ShelfScanner performs well on and classes on which it does not. To determine the difficulty of a class for ShelfScanner, we first find its strict threshold. We define the **strict threshold** of an item to be the highest threshold which results in at least one true positive. Next, we look at all the keypoints we have extracted which do not overlay an item of this class, and count the number with window distribution weight for this class exceeding the strict threshold for this class. These are keypoints that persist as false positives even with the extremely demanding strict threshold.

Using strict thresholds, we find that of the 52 item classes, 17 **easy** items can be identified with zero false positives, 8 **moderate** items require between 1 and 100 false positives, and the remaining **hard** items all require more than 100 false positives. Examples of easy item classes can be seen in Figure 5. Note that these items share the properties of good lighting, pose, and product design parity between the in vitro and in situ images. Contrasting this with the hard items in Figure 6 we see that these images vary significantly with respect to item pose, lighting, and most importantly, product design. This shows a difficulty ShelfScanner has with the dataset is likely the quality of the training data. This would underscore the need to have training data that is current with the product's design.

Figure 5: *In vitro* (top) and *in situ* (bottom) examples of easy items, which ShelfScanner can detect using a strict threshold with zero false positives.

Figure 6: *In vitro* (top) and *in situ* (bottom) examples of hard items, which ShelfScanner cannot detect with a reasonable number of false positives.

5.2. Computational performance

The average performance of ShelfScanner is reported in Table 1. Note the system is set to process every fifth frame, requiring the system to run at a minimum of 6 frames per

Component	Total time (ms)
Optical flow	70.20
SURF keypoint identification and descriptor extraction	109.74
Color histogram extraction	266.95
NIMBLE prediction	5.25
Keypoint selection	1.66
Total time	453.8

Table 1: Average time in milliseconds taken by the components of ShelfScanner.

second (FPS). As observed by the authors, the system requires this frame rate to generate a reasonably accurate mosaic. The FPS determines the average number of new keypoints in each frame, which affects the processing time of a frame. The table shows ShelfScanner can currently only process about 2 FPS, which is too slow to be considered real-time. Real-time performance could be achieved if the components were split between mosaicing tasks and detection tasks. The mosaicing portion would perform optical flow and keypoint extraction in about 180ms. Given these keypoints, the remainder of ShelfScanner could be easily run in parallel in many background threads, allowing the detection components to lag behind the current FOV without incurring a significant performance penalty.

6. Discussion and future work

ShelfScanner is a novel approach for object detection within the GroZi-120 dataset. We have shown that by leveraging the properties of grocery stores we can build a system that is capable of detecting a shopping list's items. With the aforementioned optimizations the system could be made to run in real-time. Mosaic construction proves to be the key to this approach, allowing the system to avoid processing regions of a scene twice. Future work could apply more general motion models to the camera. This would allow the system to handle a wider variety of user movement, which would increase the accuracy of our mosaics and perhaps allow for object detection based on matching keypoints via homographies.

Currently the memory cost of maintaining a large mosaic suggests that work must be done on eventually forgetting old areas of the mosaic. A quickly vanishing mosaic could also enable the system to make do with simple motion models in complex environments, as local camera motions are often well-approximated by simple motion models, where extended camera motions require more complicated motion models.

Our detection results suggest that for ShelfScanner to be truly usable, up to date product images must be used for training. Future work could incorporate text recognition to reduce the sway of product packaging.

Our timing results suggest the possibility of a real time system that can keep pace with the video stream. We intend our future work to move the system into a parallel architecture that can run in real time and take full advantage of machines with multicore processors.

7. Acknowledgements

The authors would like to thank the Winter 2010 CSE 190-A class for their helpful feedback. Further we would like to thank Florian Schroff for his assistance. This work was supported in part by DARPA Grant NBCH1080007 subaward Z931303 and NSF CAREER Grant #0448615.

References

[1] A. Adams, N. Gelfand, and K. Pulli. Viewfinder alignment. In *Eurographics*, volume 27, pages 597–606, 2008. 2

[2] L. Barrington, T. Marks, and G. Cottrell. NIMBLE: A kernel density model of saccade-based visual memory. In *CogSci*, pages 77–82, 2007. 2, 4

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. {SURF} speeded-up robust features. 110(3):346–359, June 2008. 2

[4] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981. 2

[5] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007. 1

[6] M. Irani, P. Anandan, and S. Hsu. Mosaic based representations of video sequences and their applications. In *ICCV*, page 605, Washington, DC, USA, 1995. 2

[7] D. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999. 1

[8] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, pages 674–679, San Francisco, CA, USA, 1981. 4

[9] M. Merler, C. Galleguillos, and S. Belongie. Recognizing groceries in situ using in vitro training data. In *CVPRW*, 2007. 1, 2

[10] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, pages 331–340, 2009. 4

[11] J. Shi and C. Tomasi. Good features to track. Technical report, Cornell University, 1993. 4

[12] L. B. Statistics and L. Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001. 1

[13] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *IJCV*, pages 511–518, 2001. 1