4-2014

Guiding Attention in Controlled Real-World Environments

Thomas P. Booth

Guiding Attention in Controlled Real-World Environments

by

Thomas P. Booth

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in
Computer Science

Supervised by

Dr. Reynold Bailey

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

April 2014

The thesis “Guiding Attention in Controlled Real-World Environments” by Thomas P. Booth has been examined and approved by the following Examination Committee:

Dr. Reynold Bailey
Assistant Professor
Thesis Committee Chair

Dr. Joe Geigel
Associate Professor

Warren R. Carithers
Associate Professor

Dedication

To Mom and Dad; terminat auctor opus.

Acknowledgments

This material is based on work supported be the National Science Foundation under Award No. IIS-0952631. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Abstract

Guiding Attention in Controlled Real-World Environments

Thomas P. Booth

Supervising Professor: Dr. Reynold Bailey

The ability to direct a viewer's attention has important applications in computer graphics, data visualization, image analysis, and training. Existing computer-based gaze manipulation techniques, which direct a viewer's attention about a display, have been shown to be effective for spatial learning, search task completion, and medical training applications. This work extends the concept of gaze manipulation beyond digital imagery to include controlled, real-world environments. This work addresses the main challenges in guiding attention to real-world objects: determining what object the viewer is currently paying attention to, and providing (projecting) a visual cue on a different part of the scene in order to draw the viewer's attention there. The developed system consists of a pair of eye-tracking glasses to determine the viewer's gaze location, and a projector to create the visual cue in the physical environment. The results of a user study show that the system is effective for directing a viewer's gaze in the real-world. The successful implementation has applicability in a wide range of instructional environments, including pilot training and driving simulators.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This body of work presents a means for directing a viewer's attention in a controlled, real-world environment. Gaze manipulation approaches have been shown to be effective for computer-based spatial learning [67, 3], search task completion [40], and medical training applications [35, 56]. This work focuses on a technique called Subtle Gaze Direction(SGD) [4] that combines eye tracking with subtle image-space modulations to guide a viewer's gaze in a manner that has minimal impact on the viewing experience. The modulations are only presented to the peripheral regions of the field of view and are terminated before the viewer can scrutinize them with high acuity foveal vision. Such approaches are preferred to more overt techniques that require permanent alterations to the scene to highlight areas of interest.

The goal of this work is to develop and test various approaches to extend the concept of gaze manipulation beyond digital imagery to include controlled, real-world environments. There are two main challenges to guiding attention to real-world objects: (1) determining what the viewer is currently paying attention to, and (2) providing (projecting) a visual cue on other parts of the scene to draw the viewer's attention there. In the system layout (Figure 1.1), the user wears a pair of eye-tracking glasses that allow for unrestricted head movement. The glasses are equipped with a front facing camera that captures the scene from the perspective of the subject and determines the viewer's gaze position within the scene. Gaze inference can be performed by extracting image features from this camera's video feed near the viewer's current fixation and then searching for correspondence with known objects. This process determines the location of the viewer's current fixation with

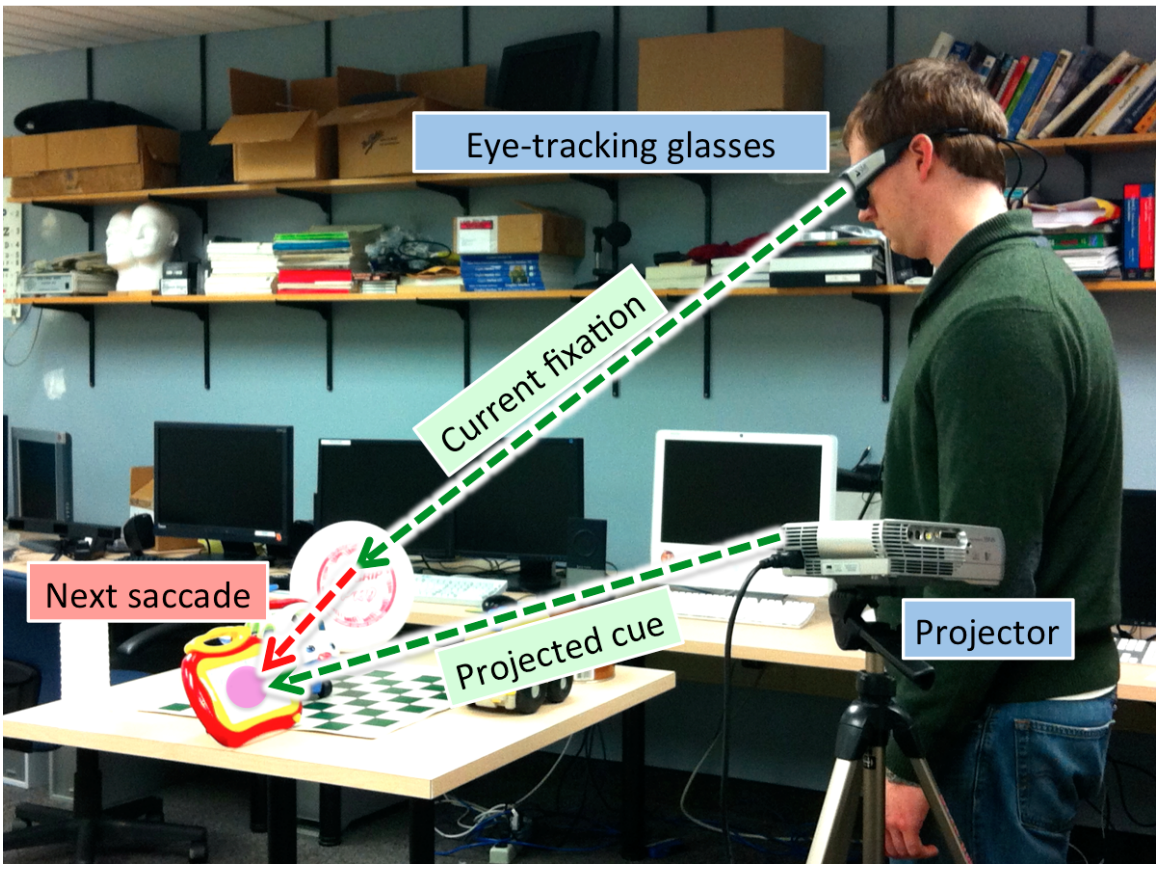

Figure 1.1: Annotated photograph of the real-world gaze manipulation setup. The current fixation is determined using a wearable eye-tracker. A projector is used to create a visual cue on another part of the scene in order to guide the viewer's gaze to that location. This process can be repeated to guide the viewer over time.

respect to observable portions of the scene. Once the subject's viewpoint has been established, a visual cue (subtle luminance change or a more overt flash) can then be projected onto another part of the scene to attract the viewer's attention. This can be repeated as necessary to guide the viewer's gaze over time.

The final system combines off-the-shelf components with special purpose software. The eye-tracking system is capable of sending real-time data streams over network socket, which is connected to a companion system. The companion is responsible for performing feature-based image processing, object detection, and generating subject stimulus. Real-time performance constraints for the companion are not unrealistic, since modern desktop computers are more than capable of performing feature-based image processing in real-time [26].

To evaluate the design methodology, a user study was conducted. The user study consisted of subjects looking freely at a scene, while imposing an arbitrary sequence through SGD. The effectiveness of system was evaluated by how precisely subjects follow the intended pattern. In the interest of gauging subtlety, each subject was interviewed for their awareness to the presence of gaze direction.

Implementing such a system has high applicability in instructional environments such as aircraft cockpits. For example, it can be used to direct novice pilots to situation-relevant instrumentation in response to an event such as equipment failure or adverse weather conditions. Elaborating on this example, instruction may include operational protocols such as takeoff and landing procedures, or the utilization of gaze data captured from veteran pilots to guide novices. Similar training can be done in driving simulators and systems for navigation aid.

Chapter 2

Background

2.1 The Human Visual System and Modern Eye-tracking

In humans, foveal vision (center of gaze) has very high acuity when compared to peripheral vision. The falloff in visual acuity as distance from the fovea increases is directly related to the distribution of the cone photoreceptors in the retina [46]. The density of cones, and hence the visual acuity, is very high in the fovea and falls off rapidly as the angle increases. The fovea itself has a diameter of $1.5mm$ and subtends an angle of approximately 2 degrees of the visual field. This difference in photoreceptor distribution is pictured in Figure 2.1. This means that at any instant, less than 0.05% of our field of view is seen in high resolution. We overcome this limitation by quickly scanning about the scene. These rapid eye movements are called saccades and the brief pauses to focus on objects within the scene are called fixations. Most saccades occur at a level below conscious awareness.

Eye-tracking provides a mechanism for monitoring where our high acuity vision gets focused in a scene or on a display. Eye-tracking systems first emerged in the late 1800s (see [24] for a review of the history of eye-tracking) and over the years various approaches have been used to track viewer gaze including magnetic coils embedded into contact lenses and electrooculograms (EOG) which attempt to determine eye-position by taking advantage of the voltage difference between the retina and cornea. Most modern eye-tracking systems are video-based — a video feed of the subject's eye is analyzed to determine the center of the pupil relative to some stable feature in the video (such as the corneal reflection). A brief calibration procedure is usually necessary to establish the mapping of the subject's

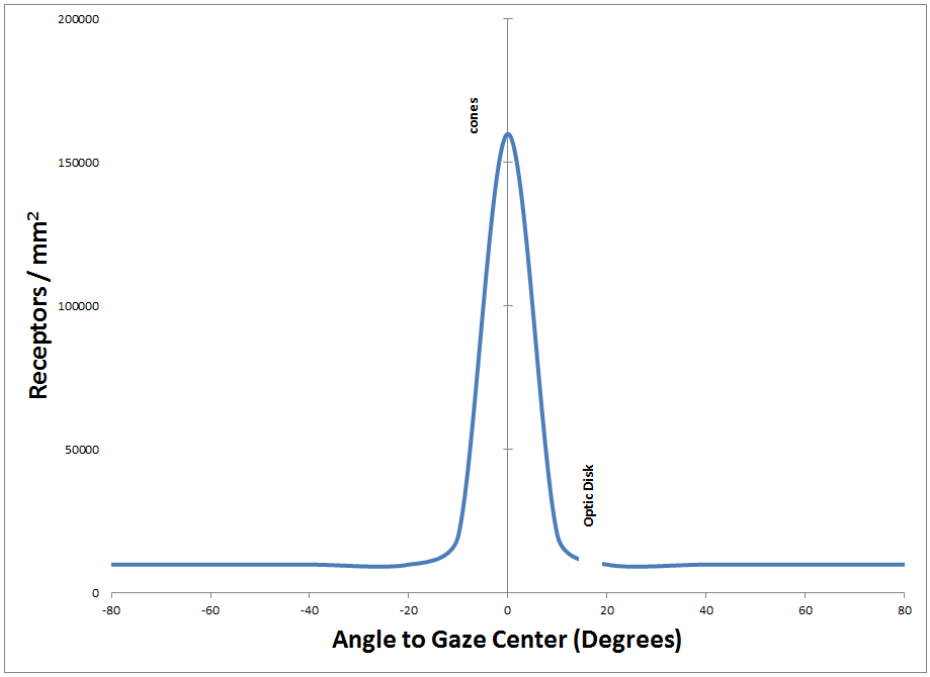

Figure 2.1: Distribution of cones in the retina. Cone concentration is highest near the center of gaze–the fovea. A rapid fall-off is evident as the relative angle increases. The optic disk blocks a region of photoreceptors, creating a blindspot [36].

eye position to locations within the scene. The accuracy of video-based eye-trackers has improved in recent years with many commercial systems reporting gaze position accuracy $< 0.5°$. Eye-tracking systems are primarily used to collect eye movement data during psychophysical experiments. This data is typically analyzed after the completion of the experiments. However during the 1980s, the benefits of real-time analysis of eye movement data were realized as eye-trackers evolved as a channel for human-computer interaction [30, 23].

Wearable eye-trackers began to emerge during the late 1990s [49, 2]. Today most of the commercial eye-tracking companies offer head-mounted eye-tracking solutions. Head-mounted eye-trackers allow researchers to capture information about the visual behavior and perceptual strategies of people who were engaged in tasks outside of the laboratory. They have already been used in a wide range of settings including driving [55], sports [8], geology [16], and mental health monitoring [65].

2.2 Subtle Gaze Direction

The finalized system is patterned after the Sutble Gaze Direction technique which was designed to guide attention in computer displays. Referring to Figure 2.2, the eye-tracker determines a known fixation in a viewing plane where there also exists a desired fixation target—a Region of Interest or ROI. For this instance in time, let $\overrightarrow{v}$ be the velocity of the current saccade, let $\overrightarrow{w}$ be the vector from the fixation to the ROI, and let θ be the angle between $\overrightarrow{v}$ and $\overrightarrow{w}$. Once it has been established that the two regions do not overlap (preempting the need for guidance), the ROI is modulated at a rate of 10 Hz.

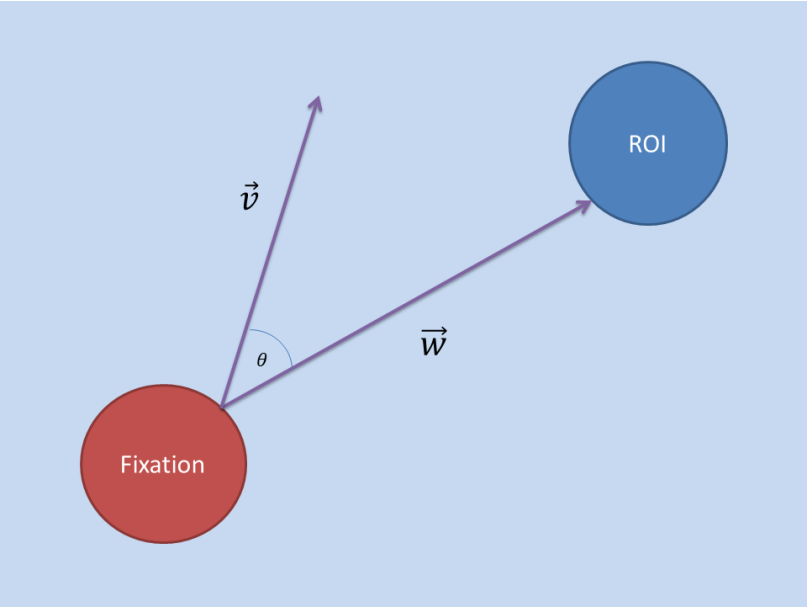

Figure 2.2: Visual demonstration of SGD

While the modulation is active, saccadic velocity is monitored using data from a real-time eye-tracking system. With known vectors $\overrightarrow{v}$ and $\overrightarrow{w}$, θ is easily calculated using the geometric equation:

$$\theta = \arccos\left(\frac{\overrightarrow{v} \cdot \overrightarrow{w}}{|\overrightarrow{v}||\overrightarrow{w}|}\right). \tag{2.1}$$

The case where θ is relatively small ($\leq 10^{\circ}$) implies that the subject's gaze is tending towards the desired fixation and the modulation should be terminated. This process can be repeated indefinitely on multiple ROIs to direct a viewer's gaze about the scene. Subtlety is achieved by leveraging the phenomenon of saccadic masking. To be more specific, the ability of a subject to perceive differences in their field of view diminishes during a saccade [15]. An example of saccadic masking as depicted over time is shown in Figure 2.3.

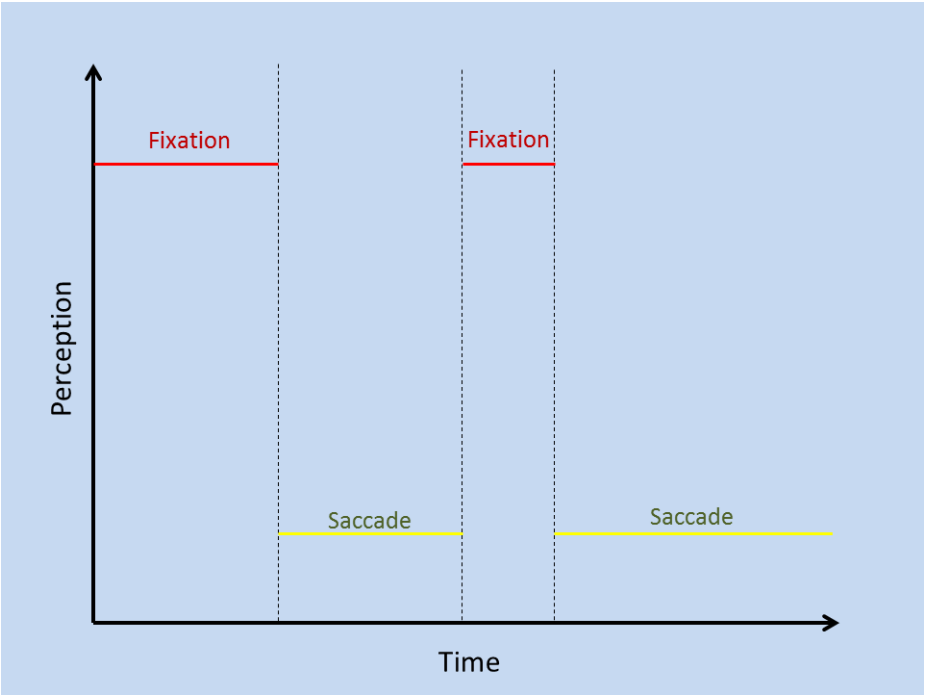

Figure 2.3: Saccadic masking

Again, the modulations are only presented to the peripheral regions of the field of view and are terminated before the viewer can scrutinize them with high acuity foveal vision. As long as the modulation is terminated while the subject is experiencing a saccadic mask, the subtlety of the technique is maintained.

2.3 Computer Vision

The fundamental goal of computer vision is to enable computers to interact with the environment using pictures or video. Focusing on object detection, most techniques for doing so rely on image analysis of static images. Although the literature for object detection is broad, a majority of algorithms fall into one of two basic categories: appearance methods (a literal interpretation of the image) and feature-based methods (decomposing regions of pixels into numeric values).

Appearance methods [57, 54, 32, 33] rely on the use of "exemplars" or images that exclusively depict the object to form a formal description. The primary concern with such methods is sensitivity to changes in lighting, viewing angle, and size of an object. The accepted solution is to use a collection of prepared images that cover expected usage scenarios. In this case, performance and accuracy are inversely proportional—a larger (or more accurate) description will be comparatively slower than a smaller (less accurate) description.

Feature-based methods [37, 5, 42, 12] decompose regions of pixels into keypoints—regions of an image with outstanding (unique) composition. The commonalities with several algorithms of this type are scale and rotation invariance—meaning that sample images depicting differences in size and orientation are not required for a successful detection. Additionally, although not immune to changes in lighting, feature-based methods do not require additional images to compensate for differences in lighting; only modifications in thresholding (chosen at runtime). To allow the detection of an object within a scene, decomposition is performed on a quality source image where the object is the principle subject; this operation builds the complete definition for the object. Next, decomposition is performed on the scene perspective (the canvas in which to detect the object). With both sets of keypoints, simple comparisons are done to determine the likelihood of a matching keypoint. An object is detected when a significant number of single object keypoints are present within the scene.

Appearance methods frequently employ multi-frame tracking within video feeds for

accuracy while feature-based methods can be utilized on single images or individual video frames with identical precision. Either type is suitable for employing SGD, however, feature-based methods do not necessarily have to only scan for objects (such as examining independent image regions for similarity). Therefore, this work will exploit feature-based methods while evaluating their effectiveness by two means: (1) using localized image analysis and (2) discrete object detection.

Chapter 3

Related Work

There has been extensive research to determine what guides viewer attention. It is well known that the pattern of eye movements depends on the viewer's intent or the task assigned [70, 22, 58]. Image content also plays a role. For example, it is natural for humans to be immediately drawn to faces or other informative regions of an image [38]. Additionally, research has shown that our gaze is drawn to regions of high local contrast or high edge density [39, 47]. Researchers continue to debate whether it is salient features or contextual information that ultimately drives attention during free viewing of static images [10, 61, 60, 7].

Jonides [25] explored the differences between voluntary and involuntary attention shifts and referred to cues which trigger involuntary eye-movements as *pull cues*. Computer based techniques for providing these pull cues are often overt. These include simulating the depth-of-field effect from traditional photography to bring different areas of an image in or out of focus or directly marking up on the image to highlight areas of interest [13, 19, 59, 68, 20]. It is also possible to direct attention in a subtle manner. For example, the Subtle Gaze Direction (SGD) technique [4] works by briefly introducing motion cues (image space modulations) to the peripheral regions of the field of view. Since the human visual system is highly sensitive to motion, these brief modulations provide excellent pull cues. To achieve subtlety, modulations are only presented to the peripheral regions of the field of view (determined by real-time eye-tracking) and are terminated before the viewer can scrutinize them with high acuity foveal vision.

Gaze manipulation has high applicability in instructional environments. Numerous studies have been conducted to understand experts' eye movements for specific tasks and

to use their fixation sequence to guide novices during training. This has been done in tasks such as aircraft inspection [53] and optic disc examinations [45]. An interesting application of this approach is in the area of medical image analysis to understand the cognitive strategies employed by experts [62, 31]. Litchfield et al. [35] [34] showed that viewing an expert's eye movements can help to improve identification of pulmonary nodules in chest x-rays. Gaze manipulation has also been used to subtly guide novices along the scanpath of an expert radiologist as they try to identify abnormalities in mammograms [56]. Results of that study reveal that novices who were guided performed significantly better than the control group (no gaze manipulation). They also continued to perform better once the training was complete and gaze manipulation was disabled.

The concept of guiding attention (pointing) in physical spaces is well established in art, photography theater and cinema. In most cases this relies on clever use of contrast and composition, direct manipulation of lights, or manual post-processing to highlight regions or characters of interest. Several augmented reality systems have also been developed to directly highlight real-world objects [69, 50, 21] or to highlight objects on a video feed of the real world [9, 17]. However, none of these systems take the viewer's gaze into account or attempt to guide attention without using overt cues. One notable exception is the work by Veas et al. [63] where they pre-process video feeds to increase the saliency of target regions while reducing the saliency of surrounding regions. They do this by adjusting the contrast of the various channels in CIELAB space to create a natural looking change to the image which increases the likelihood of the target regions being attended to. In their work, eye-tracking was only used to test if the subjects actually paid attention to target regions. Furthermore, their technique does not actively try to shift attention between multiple targets. In contrast, we plant to adopt an approach similar to the Subtle Gaze Direction technique, where eye tracking is used to monitor where the viewer is attending to in the scene in real-time. A visual cue is then projected onto another part of the scene to draw the viewer's attention. This process can be repeated to guide the viewer's gaze about the scene.

Chapter 4

Hypothesis

The hypothesis of this work is that it is possible to develop a system capable of guiding attention in the real-world. Furthermore, applying gaze guidance in the real-world should offer a noticeable increase in task-based performance of subjects.

4.1 Equipment

As mentioned before, SGD is already capable of directing gaze, but lacks a means to do so in the real-world. Extending SGD beyond a computer screen requires producing modulations in a physical space. The proposed method for providing modulations is via a projector fixed on a designated scene. Naturally, there will exist a maximum viewable area limited by the range of the projector. However, by comparison, SGD is currently limited to a particular viewing space. Therefore, current implementations of SGD and hypothetical systems will both include a maximum effective area.

Performing meaningful augmentations with a subject's perspective requires additional processing done seperately from the eye-tracker. The classical application of SGD (on a computer screen), utilizes a desktop computer for interpreting gaze velocity and providing user stimulus. The proposed system will need to provide the same level of functionality and support real-time frame decomposition. Although image analysis is clearly more intensive than the original incarnation of SGD, the companion system should still be able to provide the necessary runtime given the speed of modern desktop computers.

4.2 Performance

SGD relies on the speed of modern eye-trackers for its subtlety. Specifically, when a subject is experiencing saccadic masking while they transition between two gaze points, the modulation terminates. This implies that the speed of the gaze interpolation must be equal to or less than the saccad flight time. Second, the ideal performance of object tracking would be done frame-by-frame or every other frame. The eye-tracker chosen for this experiment operates at 24 frames per second; implying the total computation time must be less than 83 ms (for alternating frames) and ideally 41 ms (for perfect frame updating).

4.3 Subject or Object Movement

While a truly dynamic augmentation would be desirable, it is beyond the scope of this work. The principal issue regarding object movement is tracking movement in relation to the user's current fixation. Once an object moves, the positioning of the modulation within the projection field changes; therefore, object targeting information of the projector must be updated. This processing would need to be performed twice—once for the subject and a second time for the perspective of the projector. There must also be considerations for calibrating the projector viewpoint and the secondary camera viewpoint; so that the observable boundaries (or effective area) are consistent. With support from the aforementioned analysis, object movement will be left as future work.

Subject movement is largely expected—particularly when examining multiple objects within a scene. Object detection is immune to subject movement because there is no *a priori* knowledge of object location. The object must be first detected, then considered for bounding values, and finally compared against the current gaze velocity. However, excessive movement is prohibited because of the limited frame rate of the eye-tracking perspective camera. Fast head movements will render the scene captures blurry and inhibit the object detection process.

In summary, the overall assumption is that objects will remain stationary while the subject is allowed controlled head movement for comfortable viewing of the scene.

Chapter 5

Design

As discussed earlier, the two main challenges to guiding attention in real world environments are (1) determining what the viewer is currently paying attention to, and (2) projecting visual cues on other parts of the scene to draw the viewer's attention. The system combines the use of eye-tracking glasses, a standard projector, and a high-resolution source images. The source information consists of either an image of the scene image from the perspective of the projector, or a collection of individual objects present in the scene. The eye-tracking system computes the fixation position within the scene that the viewer is attending to. Keypoint and descriptor extraction is performed on the sourcing image(s) prior to system runtime and on individual scene frames as they occur. Comparisons are performed between scene and source descriptors to infer gaze location. This is an important step in the process as it contributes to not attempting to guide a viewer objects they are already attending to. Once the current fixation is determined, viewer attention can then be guided by projecting a visual cue onto another part of the scene.

The following is a list of design goals:

- The system should perform in real-time.
- The system should allow for unrestricted movement of the viewer as long as their fixations remain within the region covered by the source image.
- The system must be robust even under ill-conditioned and ill-posed situations such as track-loss or when the viewer looks away from the scene.

The hypothesis is feasible so long as the following assumptions are made:

- The system will be used in a visually stable environment (i.e. unchanging external lighting and no moving objects).
- The objects in the scene are non-transparent and well defined in terms of color, contrast, size, shape and texture.

5.1 Runtime Analysis

To ensure real-time performance, preliminary testing was done to estimate the actual time necessary to discover keypoints, compute descriptors, and perform matching on available equipment. The OpenCV library [6] provides a wide array of image processing algorithms and was used in implementation and testing. Note that while each algorithm has a dedicated parameter set, OpenCV defaults were used in testing. A complete listing of equipment, testing parameters, and sample code can be found in Appendix A.

Figure 5.1: Test Scene Image

5.1.1 Keypoint Detection

While the exact technique varies between algorithms, keypoint detection methods all perform the same function—that of identifying regions of a static image that are significant. Testing begins by comparing the runtime of these operations implemented entirely in a CPU-based approach on an image with dimensions 1280x960 (taken with the eye-tracker scene camera, shown in Figure 5.1). The results of this test are summarized in Table 5.1.

Algorithm	Time (s)	Keypoints
SURF	0.2924300	400
FAST	0.0122027	1906
SIFT	2.5448000	713
STAR	0.1743090	139
BRISK	0.0573439	182
ORB	0.0793547	500
MSER	0.4560330	123

Table 5.1: Keypoint Detection Runtimes

From the results, FAST[51] provides the most keypoints and shortest runtime. Additionally, the number of keypoints extracted via FAST would suggest that more restrictive thresholding would be adequate and leverage even greater performance. The dataset generated by FAST will therefore be used in subsequent stages of testing.

5.1.2 Descriptor Extraction

While FAST is an effective keypoint *detection* algorithm, it does not include descriptor *extraction*. For this reason, FAST will be omitted from the test results (as will STAR and MSER, by the same rationale). Referring to the result from Section 5.1.1, descriptor extraction testing is performed on a dataset of roughly 1900 keypoints. Table 5.2 shows the runtime performance of various extraction algorithms.

Algorithm	Time (s)
SURF	0.0910057
SIFT	2.4128
BRISK	0.0405125
ORB	0.0481414
FREAK	0.0581075

Table 5.2: Descriptor Extraction Runtimes

The results suggest that BRISK [28], FREAK [1], or ORB [52] would be suitable extractors in the interest of speed.

5.1.3 Descriptor Matching

While an item-by-item (or brute force) comparison of descriptors is a simple operation, such an algorithm is not well suited to real-time performance. Included in OpenCV is an implementation of the Fast Library for Approximate Nearest Neighbors (FLANN) [43] for fast matching of binary features (between scene and source) [44].

Descriptor Count	Time (s)
1906	0.221049

Table 5.3: Descriptor Matching Runtime

The initial result in Table 5.3 is well over the established maximum of 83 ms (see section 4.2). A simple optimization would to more aggressively constrain keypoint detection inorder to decrease the dataset size. Instead of the OpenCV default of $threshold = 1$, a new dataset is created using FAST with $threshold = 30$. This change reduces the total number of descriptors from roughly 1900 to approximately 249. An updated runtime for FAST is recorded in Table 5.6 and the result of this reduction is shown in Table 5.4 (below).

Descriptor Count	Time (s)
249	0.0555692

Table 5.4: Descriptor Matching Runtime (Threshold Restricted)

While this result is promising, additional optimization can be had by limiting the relative distance (or quantitative difference) between two keypoints during the matching procedure. By decreasing the maximum distance, matches that are least likely to be accurate are discarded. The result of a reduced dataset while enforcing a maximum distance is shown in Table 5.5.

Descriptor Count	Maximum Distance	Time (s)
249	100	0.0508773

Table 5.5: Descriptor Matching Runtime (Threshold Restricted and Distance Limiting)

Elaborating on the effect of distance thresholding, the distance between two basic descriptors—D can be defined as:

$$D = \left(\sum_{i=1}^{64} (x_i - y_i)^2 \right)^{1/2} \tag{5.1}$$

where x and y are independent descriptor sets of size 8 x 8. Given that a descriptor is extracted on 8-bit grayscale images, the maximum distance for a given descriptor element is 255, and for a descriptor block, 2040. In using the maximum distance in Table 5.5, the

average difference per descriptor block element would be $\sim$12.

5.1.4 Total CPU Runtime

Table 5.6 (below) shows the total time required for each optimized stage of the frame processing pipeline.

Pipeline Stage	Time (s)	%Total
Keypoint Detection	0.00878618	11.9
Descriptor Extraction	0.0140017	19.0
Descriptor Matching	0.0508773	69.1
Total:	0.07366518	

Table 5.6: Frame Execution Time

The runtime breakdown clearly shows descriptor matching as the time-critical portion of the pipeline, as it constitutes 69.1% of the total. From this result, it is concluded that an entirely CPU-based approach will be limited to every-other-frame as it lies above 42 ms and less than 83 ms.

5.1.5 GPU Acceleration

OpenCV provides GPU-based implementations for a subset of common algorithms; which includes a brute-force descriptor matcher. Borrowing the exact parameters for the testing performed in Section 5.1.3, Table 5.7 (below) outlines the runtime.

Descriptor Count	Time (s)
249	0.00210619

Table 5.7: Descriptor Matching Runtime (GPU Accelerated)

Leveraging GPU acceleration nets a significant speed-up of **24** for descriptor matching.

5.1.6 Final Optimized Runtime

Table 5.8 (below) outlines the optimized runtime of each major processing pipeline stage while incorporating GPU acceleration.

Pipeline Stage	Time (s)	%Total
Keypoint Detection	0.00878618	35.3
Descriptor Extraction	0.0140017	56.2
Descriptor Matching	0.00210619	8.5
Total:	0.02489407	

Table 5.8: Frame Execution Time (GPU Accelerated)

Given the results, frame decomposition time will not be the limiting factor in enabling real-time performance as each frame will require less than 42 ms of computation time.

5.2 System Overview

Figure 5.2 shows the architecture for the real-world gaze manipulation framework. The eye tracker utilized is a SensoMotoric Instruments (SMI) Eye Tracking Glasses 2.0. The glasses are equipped with a front facing camera (scene camera) that captures the scene that the viewer is looking at as well as two rear facing cameras that capture binocular eye movements (eye cameras). The resolution of the scene camera is 1280 x 960 @ 24p and the field of view is 60° horizontal and 46° vertical. Gaze position within the scene is computed using SMI's proprietary eye-tracking software at a rate of 60 Hz. The eye tracker is connected to a dedicated laptop which computes the viewer's fixations on the scene camera frames in real time. The laptop is powered by a dual core processor with 4 GB RAM. The compressed video and fixation data are streamed over the network to a desktop for further processing. The desktop is equipped with 24 GB RAM, dual hexa-core processors and a GTX 590 GPU. A standard XGA (1024 x 768) projector completes the hardware for our system.

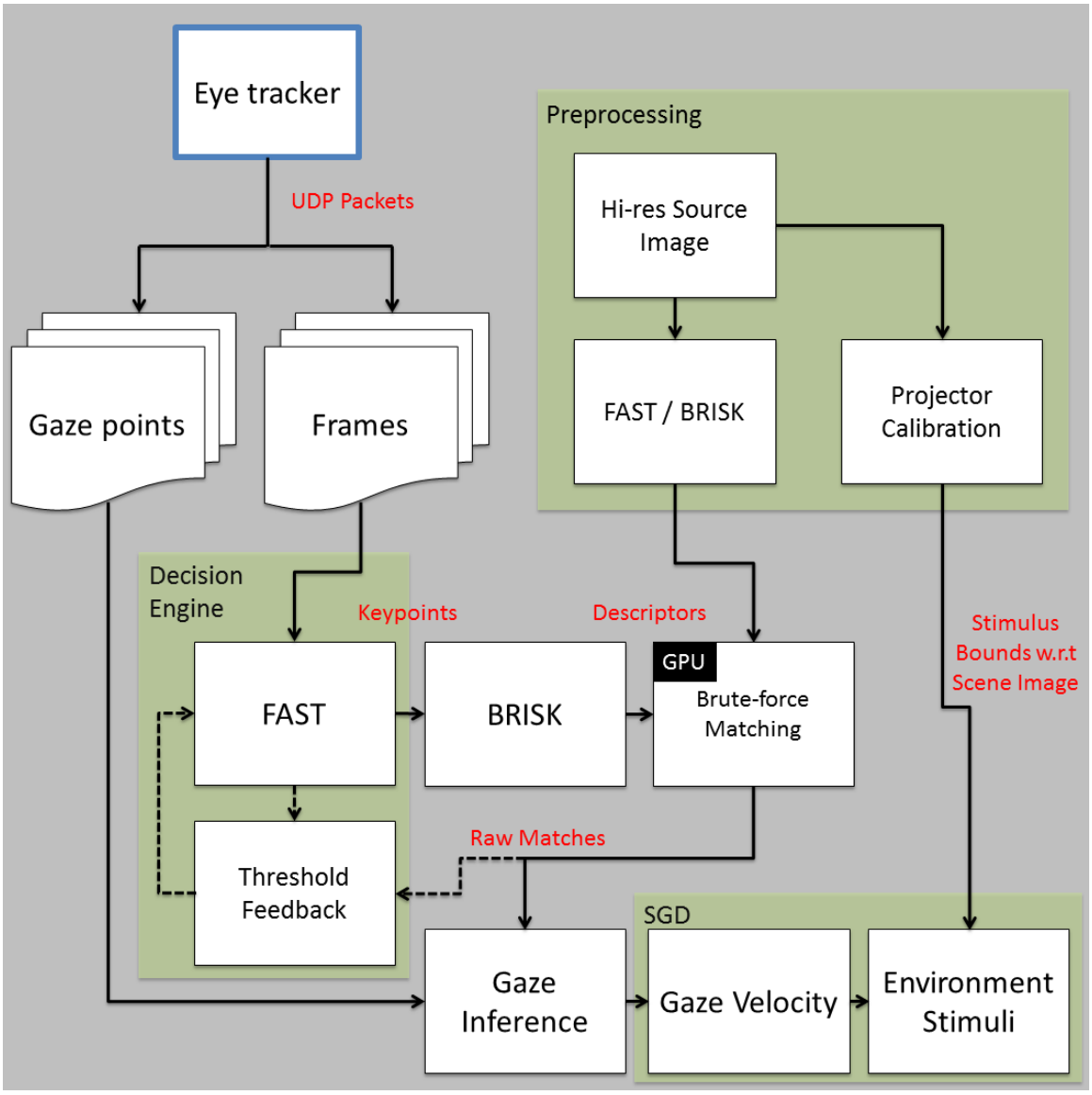

Figure 5.2: System architecture and data flow.

Descriptors of the source image are computed prior to system runtime to compare against incoming frames. The source image is also used in calibrating the limited range covered by the projector; referred to as the projection swath. The eye tracker system streams gaze positions and scene camera frames in real time over a point-to-point connection via UDP socket. These gaze points and the compressed scene frames are immediately processed by the FAST, then by BRISK; finishing the image decomposition. The thresholding values are set at the library defaults and optionally augmented by a decision engine to obtain optimum results for the given scene under varying lighting conditions and camera exposure (see Section 5.2.1). Descriptor matching is done in brute-force fashion between the scene and the source image. The resulting matching points are used by a multi-threaded CPU routine to infer where the viewer is attending to in the source image. A routine that implements Subtle Gaze Direction(SGD) uses the inferred gaze position to calculate gaze velocity within the confines of the projection swath and determines the appropriate stimuli.

The following sections detail the considerations taken in prominent stages of the pipeline.

5.2.1 Decision Engine

Most, if not all image decomposition techniques include means for adjusting sensitivity under varying image compositions. Thresholding parameters compensate for real-world events such as: lighting, camera exposure, and image sharpness. While it would be ideal to choose a constant “average” value for system runtime, the aforementioned circumstances happen frequently enough (even in well-constrained environments) to make a single thresholding value unfeasible for this particular application. For this reason, a simple feedback system was created to automatically adjust thresholding to changes in environment. Figure 5.3 (an inlay of ”Decision Engine” in Figure 5.2) depicts the dataflow mechanism for adjusting threshold parameters.

Feedback is based on considering the number of successful matches from previous frames, current threshold values, and the number of keypoints detected in the current frame.

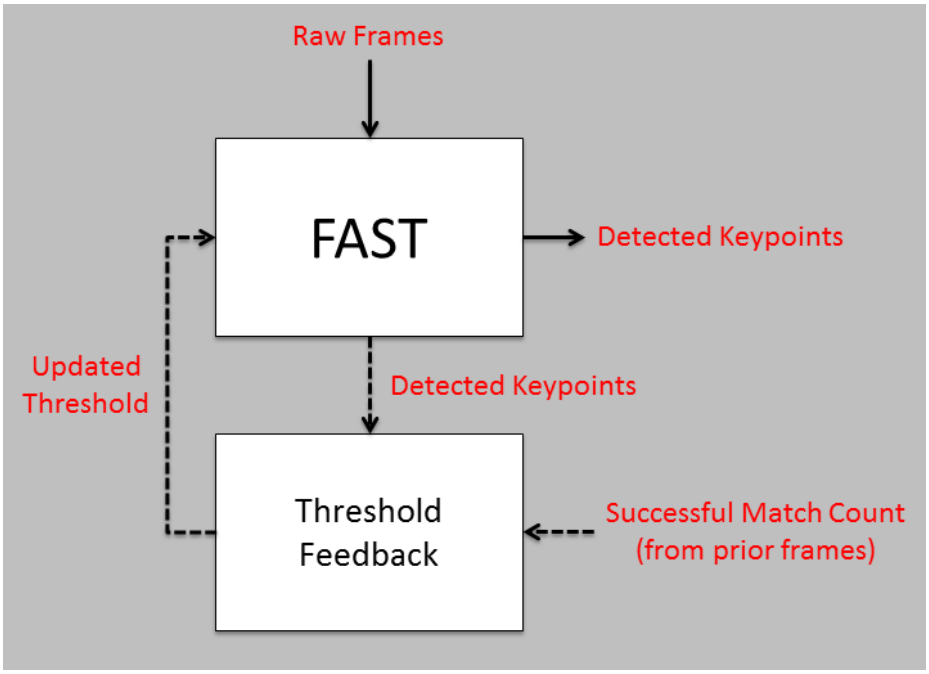

Figure 5.3: Data Flow for Decision Engine

The decision engine is designed to account for several conditions:

- An insufficient number of keypoints were detected in the current frame. Decrease detection threshold and optionally reevaluate the frame under FAST.
- No acceptable matches were returned with the prior frame. Decrease detection threshold or modify the number of octave layers.
- Multiple acceptable matches were returned with the prior frame. Increase detection threshold or decrease the number of octave layers.
- A single match was obtained. Maintain detection threshold.

Applying these specific events to state control builds the definition for the decision engine. Figure 5.4 is an illustration of the described program flow.

The internal bounds (such as those for threshold and octave layers) were chosen based on test procedures similar to those performed in Section 5.1. In testing, images like those

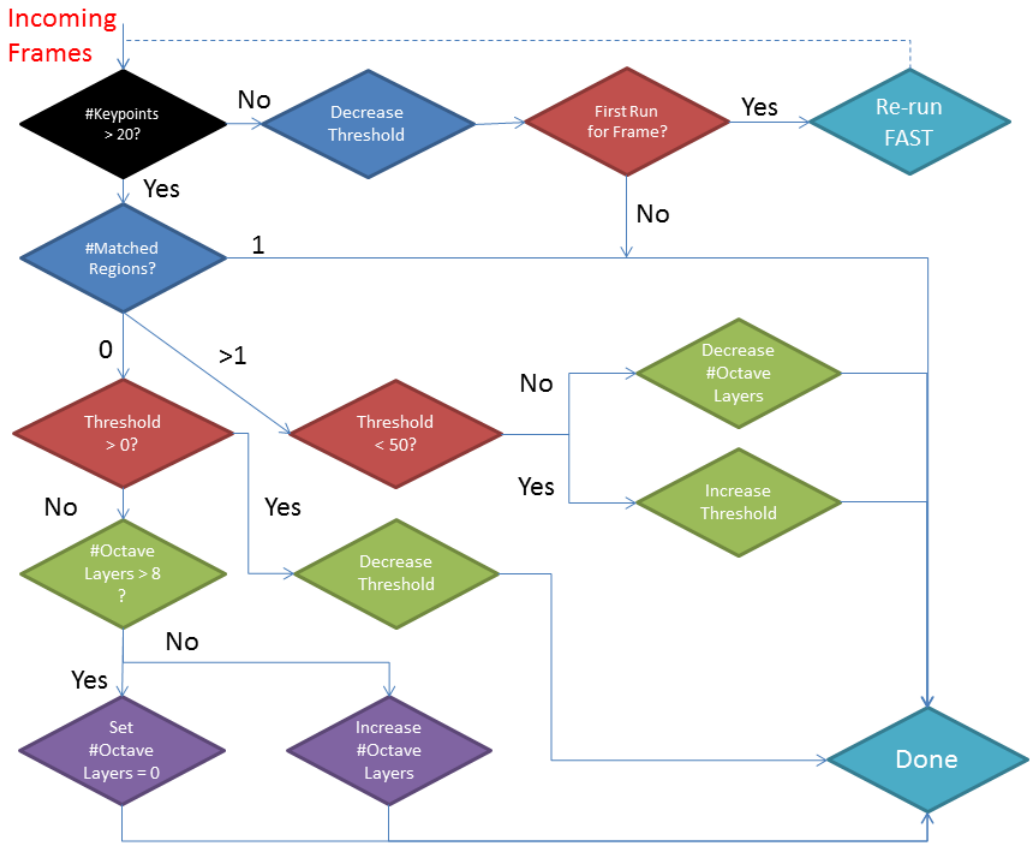

Figure 5.4: Program Flow for Decision Engine

depicted in Figure 5.1 did not return enough discrete keypoints for region matching when constrained to a $threshold$ over 50. Therefore, thresholds over 50 will be restricted from use in the decision engine. In addition, modifying the number octave layers was useful in increasing the likelihood of a match when modulating $threshold$ alone proved insufficient. Finally, matches were most likely to occur when the number of octave layers remained between 0 and 8.

5.2.2 Fixation Inference

Once descriptors for the current frame have been identified, they are matched with the pre-computed descriptors from the source image as shown in Figure 5.6. To filter the matching

results, a mask is applied about the fixation point radially until 5% of the total number of descriptors in the source image have been found. This limitation provides sufficient information to infer the fixation position while minimizing computation costs incurred from a larger comparison dataset. Cook's Distance measure [11] is used to eliminate incorrectly matched points from this subset of descriptors. Cook's Distance performs dimensionless weighting of all values in a set, with respect to the mean of that set. In this application, it is expected that the positional average of the subset of descriptors obtained in the scene image will be close to the fixation point (in the Cartesian sense). This means that the matched points with higher Cook's Distance values are more likely to be outliers. Once the outliers have been eliminated, a geometric system of equations is used to infer the fixation on the source image.

The systems of equations used to infer gaze are based on the observation that matching keypoints should originate in a similar direction; given that both scene and source image share approximately the same perspective. Figure 5.5 illustrates the geometry of the gaze inference mechanism.

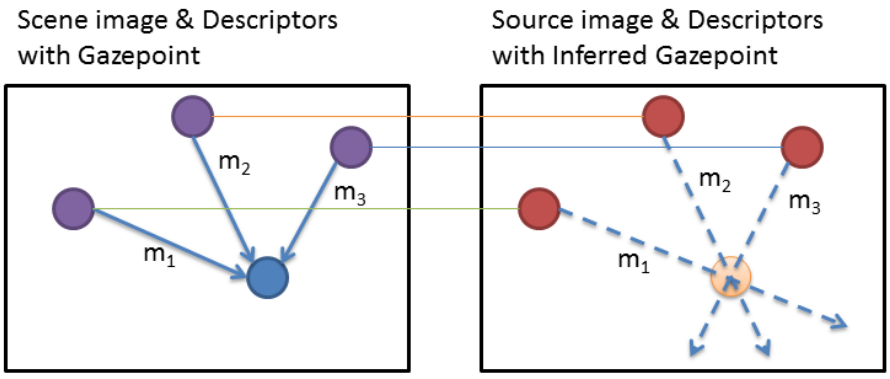

Figure 5.5: Scene image descriptors (purple, left box) lie along direction m with respect to the gazepoint. The matching source image descriptors (red, right box) inherit the direction vector of their respective matches. The inferred gaze position (orange, right box) is determined by computing the intersection between the collection of direction vectors.

The caveat with inferring gaze with the described method is that it is sensitive to rotational transformations—meaning that any significant rotation ($> 15°$) by the subject will cause the inferred position to be incorrect. However, given the controlled environment originally hypothesized, this limitation will be largely avoided. In Section 8.1, suggestions are provided for eliminating this restriction.

As an example of real-world operation, Figure 5.6 depicts a demonstration of the gaze inference mechanism; including the incorporation of Cook's Distance to eliminate suspected outliers. Note, that while the immediate region around the gaze fixation was computed considered in the matching, there are additional (unlabeled) descriptors detected within the scene.

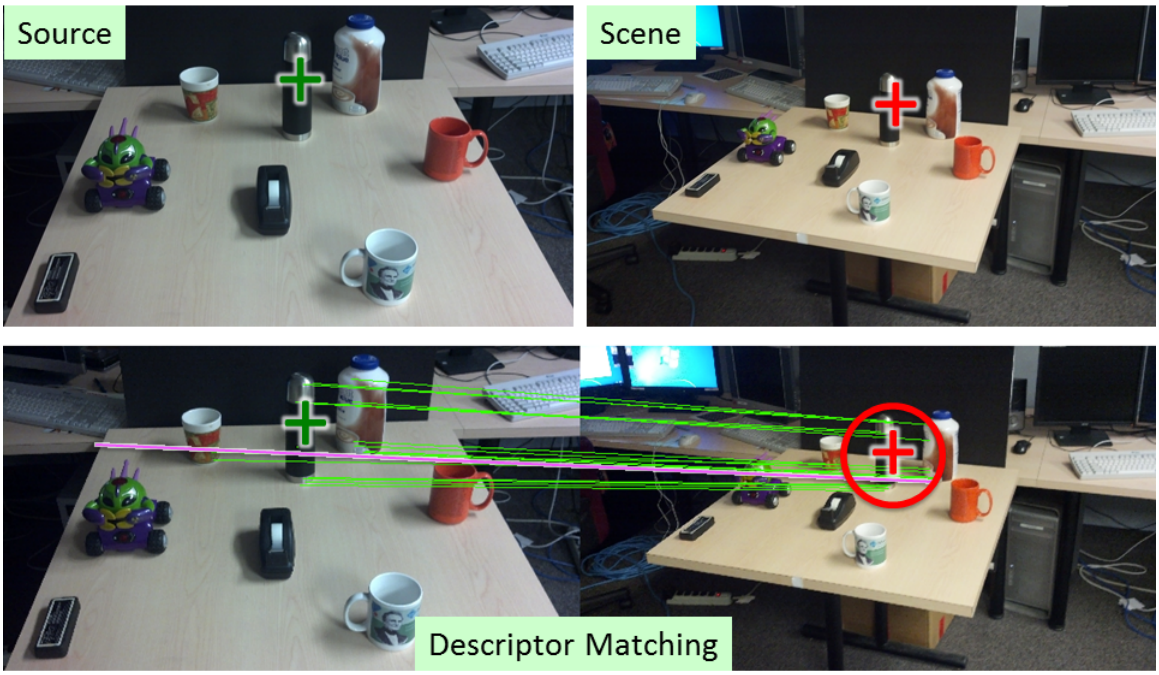

Figure 5.6: Keypoint detection, descriptor extraction, and matching with inferred fixation. Source image (top left), scene image (top right), and 5% of the resulting descriptor matches (bottom). Correct matches are shown in green and an incorrect match (outlier) is highlighted in magenta; current fixation is shown as a red cross; inferred fixation is shown as a green cross; the red circle shows the area in which the descriptors are matched.

A derivation for the system of equations used can be found in Appendix B.

5.2.3 Projector Calibration

Projector calibration consists of determining the precise area the projector is capable of illuminating and providing transformations between scene image and projector perspectives. This area of illumination will be referred to as the ”projector swath”. To determine the swath in a controlled environment, simply provide the projector with a completely white image and project it onto the scene viewable by the subject. Then, using the source image, precisely mark (using a dedicated user-interface) the bounds of the swath. Figure 5.7 (below) visualizes this operation. When applying stimuli back to the scene, a simple homog-

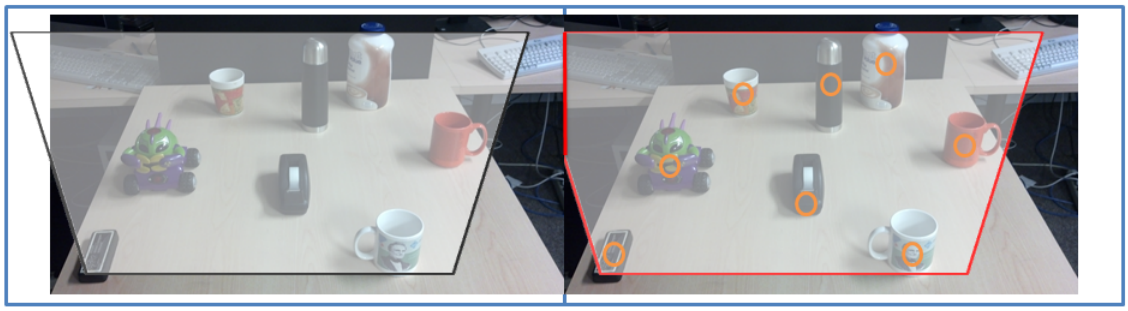

Figure 5.7: An example of projector overlap with respect to the scene image. In the left pane, the projector swath exceeds the scene image bounds but covers all the objects. In the right pane, the projection swath is clipped at the left-most source image boundary. These bounds provide the Cartesian limits for applying real-world stimulus. The orange markup (provided by the user, right only) indicates the location of gaze-guided objects.

raphy transform is performed to relate the gaze-guided objects to the Cartesian coordinates of the projection swath. This is necessary when the projection horizon is not orthogonal to the projection medium. In Figure 5.7, the projector is placed away from the projection medium (a table) and at an angle. To exclude the necessity for a homography transform, the projector would need to be mounted above the table and pointed directly at the floor.

5.2.4 Stability Considerations

Occasionally, situations may arise that lead to inaccurate results or an inability to compute the fixation position on the source image. The most common is a track-loss (i.e. the eye-tracker is not able to detect the pupil due to blinks or extreme pupil position). In this case the scene video frames will have no accompanying gaze information making fixation inference impossible. When this occurs, the system is will discard incoming frames and resume once gaze positions are again available. Also, if the viewer happens to look away from the scene, then the inferred gaze position will naturally be incorrect. Detection of non-similar scenes is done by keeping track of the number of matched descriptors from frame-to-frame. If the number of matches changes drastically (indicating dissimilar scenes) then the system ignores the matching result. Finally, it is possible (although highly improbable) for the least squares solver not to converge. This can only happen in cases where there are fewer than two descriptors present (i.e. fixation occurs in a large uniform region of the scene) or if all descriptors are co-linear. If either of these cases are encountered, the system will reuse the result of the previous frame. These contingencies collectively ensure that the system is both accurate and stable.

Chapter 6

Implementation

Before any real-time implementation was created (or even attempted), a great deal of time was spent on developing and testing prototypes of the gaze inference mechanism. Using MATLAB and VLFeat [64], the prototyping stage began by experimenting with various Computer Vision algorithms for application to real-world gaze manipulation. Once a prototype was completed, the transition was made to C++ and OpenCV [6].

6.1 Prototyping

Initially focusing on accuracy, the prototype leveraged SIFT [37] for keypoint detection and descriptor extraction, and a brute-force matching algorithm. VLFeat, a popular C-based Computer Vision library with interfaces for MATLAB was used. The matching algorithm included in VLFeat scores matches based on the Euclidean distance between two unique descriptor matrices.

The first test conducted tested the efficacy of Computer Vision in the typical operational environment expected for the final system. A sample image, (in this case, a static image of a flight simulation), is projected onto a screen. Using an off-the-shelf webcam, capture images of the projection and attempt descriptor matching between the projected image and the original. Then, apply a gaze point randomly to the captured scene image and evaluate the correctness of the gaze inference algorithm. Figure 6.1 shows the initial stage of the prototype.

Figure 6.1: The image on the left is a static image of a flight simulator while the image on the right is the left image projected and photographed using a webcam. The point on the left image (in green) indicates the inferred gaze point, while the point on the right image (in red) shows the arbitrarily chosen gazepoint. Multi-colored lines in the left image indicate data points used in computing the inferred fixation.

The obvious problem present in Figure 6.1 is the incorrect inferred fixation. A simple observation is the inclusion of a single descriptor on the far right of the image—this particular descriptor is an incorrect match. To circumvent this, Cook's Distance measure [11] is used to exclude points that are distant from the average location of the descriptor cloud (in the Cartesian sense). The result of applying Cook's Distance is present in Figure 6.2.

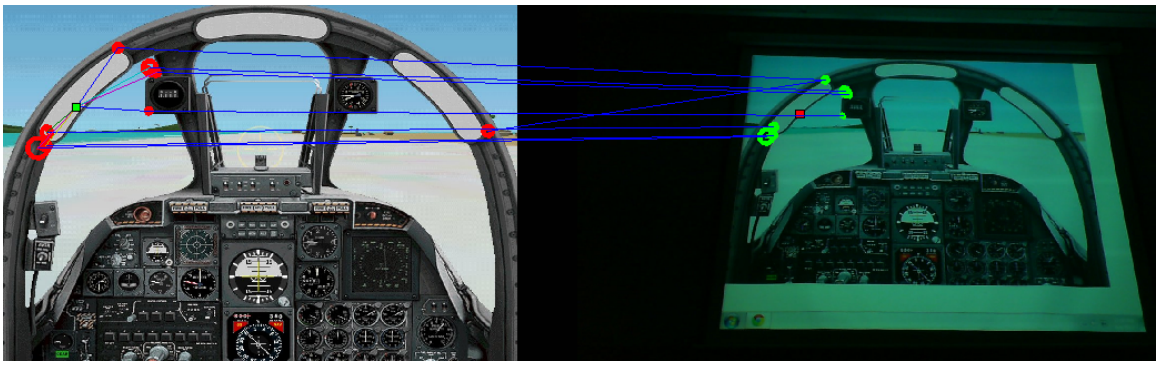

Figure 6.2: Compared to the result in Figure 6.1, this result correctly excludes the mismatched descriptor and correctly infers the fixation point.

To further examine the prototype, the same webcam is configured to capture images in

a stream, while the process of image decomposition and gaze inference are repeated. The resulting program exhibited performance similar to static test and served as the basis for the final implementation.

All source code used in development can be found in Appendix C.

6.2 Stand Alone Implementation

Choosing a platform for implementing the prototype was dependent on the supported application of the API provided with SMI's Eye-tracking Glasses (ETG). With the ETG, the only platform supported is Windows (x86) and the only language is C. By contrast, advanced functionality of OpenCV (for the purposes of this work, Feature Detection and GPGPU acceleration) is only available in C++. While the ETG API lacks an Object Oriented approach, it can still be invoked via C++. For this reason, C++ was chosen as the primary language of the implementation.

In the effort of providing a high-functioning GUI, C# was used to write the application entry point and console interpreter. With the introduction of C#, rather than utilize an external, C++ based library for UI functionality, (such as SFML [18] or Qt [14]), Windows Forms [41] were used.

Due to inclusion of two languages, the system is strictly divided into two components: (1) the UI (written in C#), and (2), the unmanaged DLL (written in C++). Again, this division is primarily done to utilize the integrated UI components in .NET and C#. A side effect of this division is a certain autonomy that is granted to the DLL by restricting UI dataflow from being processed by unmanaged code. With this configuration, the UI will pass asynchronous commands to the DLL and no interaction occurs in reverse.

Chapter 7

Analysis

7.1 User Study

The goal of the user study was to test the effectiveness of the developed real-world gaze manipulation technique. Participants viewed a simple scene consisting of eight objects. The intended viewing order was not prescribed. After viewing the scene for a short period of time, an attempt was made to guide their attention through six sequences of objects in the scene. The relevant objects were highlighted by projecting a brief luminance modulation using the projector.

The modulations were constructed by alternately blending some amount of black, then some amount of white. The rate at which the blend is modulated is 10Hz. A Gaussian falloff is used to soften the edges of the modulated regions. In the viewing configuration, the modulation diameter on the physical objects ranged from approximately 2 centimeters to 4 centimeters (depending on distance from the projector).

Figure 7.1 shows the source image of the scene used in the user study. The objects in the scene were non-transparent and well defined in terms of color, contrast, size, shape and texture. The objects were highlighted in a randomized order to minimize the introduction of learning effects.

Twenty participants (16 males, 4 females) between the ages of 18 and 29 volunteered for the user study. They all had normal, or corrected-to-normal, vision and were naïve to the purpose of the experiment—they were simply instructed to look at the scene. Each participant underwent a brief calibration procedure, away from the scene, to ensure proper

Figure 7.1: High resolution source image of scene used in the pilot study.

eye-tracking. Six randomly generated viewing sequences were presented to each participant with a 10 second gap between sequences. Each sequence consisted of 8 objects. Between sequences the participants were told to reposition themselves however they saw fit but to keep the scene visible. This ensured that the data collected across all subjects covered a wide range of vantage points. The entire experiment (including calibration) for each participant was less than 10 minutes. The viewer's gaze positions within the scene as well as the individual scene camera frames were recorded during the experiment. Data from one participant was excluded due to an extended period of track loss.

7.2 Results

It was determined that a robust mechanism to compare the intended viewing sequence with the actual viewing sequence of each participant was needed. The actual viewing sequence is extracted from the eye-tracking data by identifying the first fixation that occurs after the onset of the visual cue. Levenshtein distance [29] provides an appropriate measure to compare distances between ordered sequences, such as those recorded during the experiment. Labels ranging from A through H are assigned to the eight objects in the scene as shown

	Sequence 1	Sequence 2	Sequence 3	Sequence 4	Sequence 5	Sequence 6	All Sequences
Average	0.58	0.53	0.89	1.11	1.05	1.05	**0.85**
Standard deviation	1.12	0.90	1.15	1.70	1.51	1.08	**1.24**

Figure 7.2: Summary of Levenshtein distances across all sequences for all participants.

in Figure 7.4 (inset). These labels used to compare the subject's viewing sequence with the intended viewing sequence. Suppose for the eight objects in the scene that the correct viewing order is [ABCDEFGH]. A Levenshtein distance of 0 would indicate no difference (perfect subject ordering match), whereas a distance of 8 would indicate the maximum difference (subject was unable to match even one object during the ordering).

Figure 7.2 shows the average and standard deviation of the Levenshtein distance for the six sequences for all participants. The similar averages and standard deviation values indicate that participant performance remains consistent over time. The average of the Levenshtein distance across all sequences for all participants is 0.85 (recall that the Levenshtein distance measure for this study ranges from 0 to 8). This means that less than one error per sequence can be expected. The histogram in Figure 7.3 shows the distribution of Levenshtein distance for all sequences. Notice that it is skewed to the left indicating that a large number distance measures are close to zero. Only 18% of the trials had more than one sequence error and the error rate falls off rapidly.

It was also observed that the response time between the onset of the modulation and the subsequent fixation on the object was approximately 0.5 seconds. This is consistent with what was observed in the Subtle Gaze Direction paper for digital images.

Overall these results indicate that the system is indeed effective at guiding attention in simple controlled real-world environments. Figure 7.4 shows a representative frame from the scene camera video that was captured from one subject for one of the sequences; it is overlaid with the subject's scanpath. The numbered red circles indicate the order of the fixations that occurred during the presentation of the sequence and the size of the circle

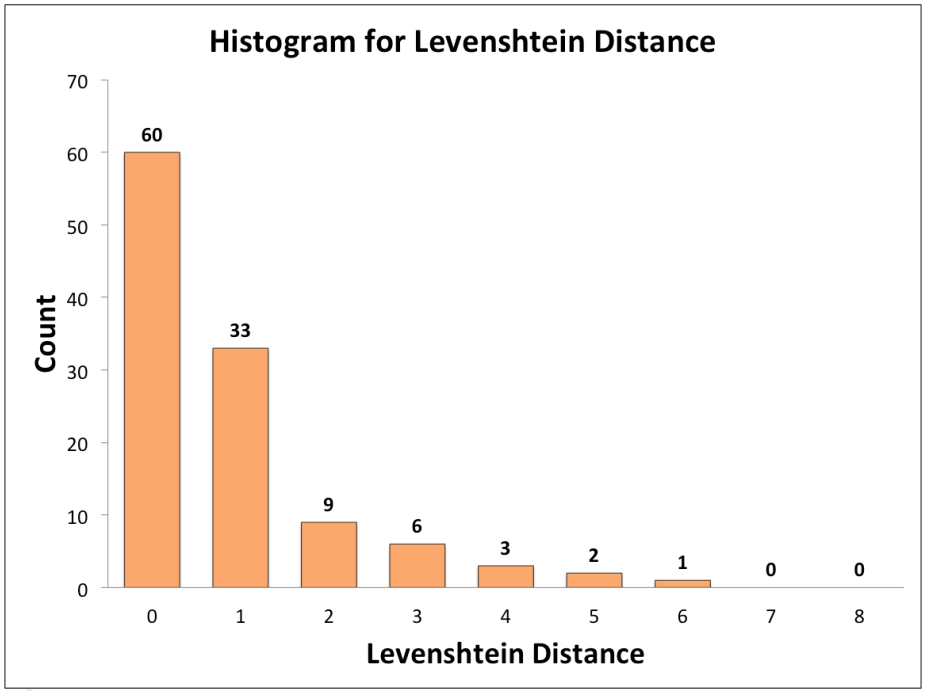

Figure 7.3: Histogram of Levenshtein distances across all sequences for all participants.

indicates the duration of the fixation. In this particular example there was a perfect match between the target sequence and the actual sequence.

To demonstrate the usefulness of real-world gaze manipulation, the system was evaluated in a parts retrieval experiment. Parts from a toy building set were arranged on a table in piles (see Figure 7.5) and six subjects were asked to retrieve 10 parts. Three subjects were given a sheet of paper with photos of the 10 parts (control) and three subjects were instructed that gaze guidance would be provided. The results of this experiment are shown in Figure 7.5. As expected, subjects in the control group had to shift their attention back and forth between the printed sheet and the parts on the table. They also had to develop a strategy for keeping track of which parts were already found. This resulted in longer completion times and greater likelihood of error. On the other hand, the gaze guided group performed much better in terms of completion time and error rate due to reduced cognitive load.

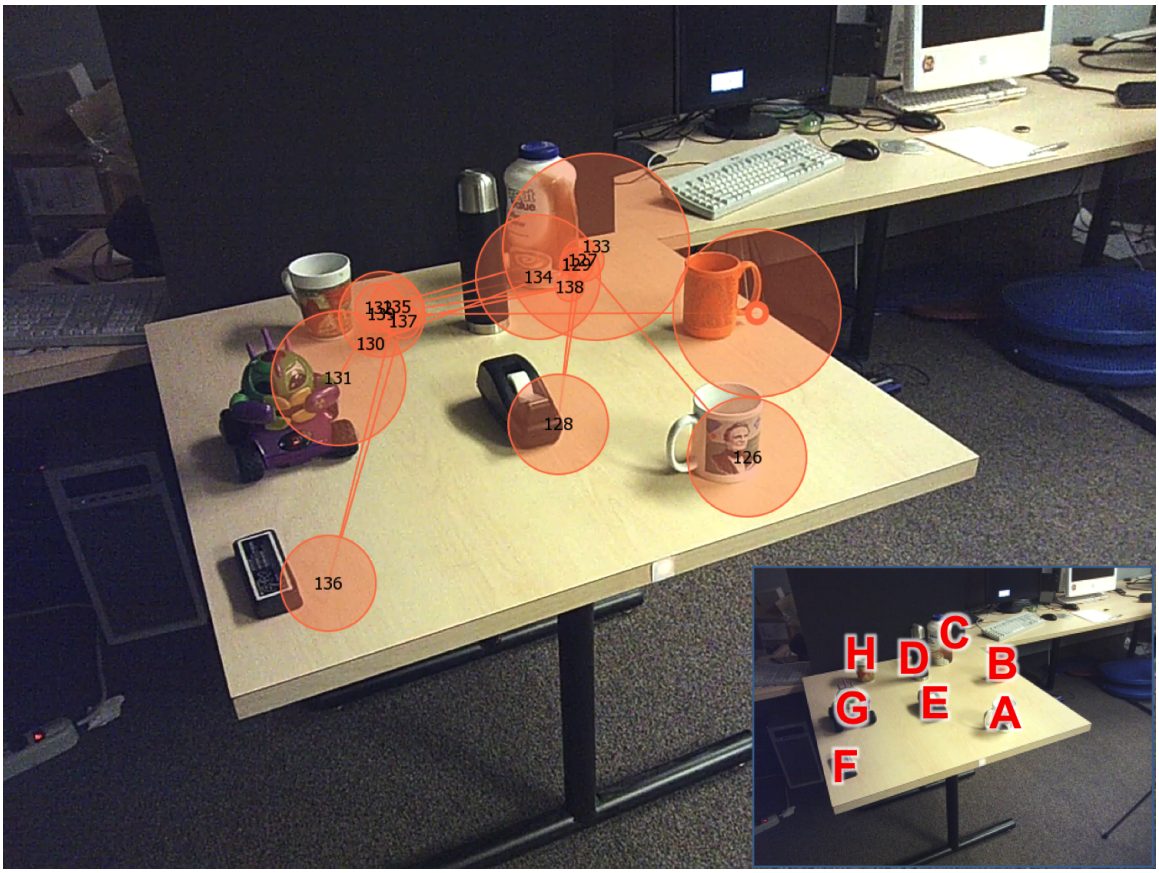

Figure 7.4: Representative images showing the real-world gaze-guided scanpath of a single subject. The numbered red circles indicate the order of the fixations that occurred during the presentation of the sequence and the size of the circle indicates the duration of the fixation. Inset show the labels assigned to the various objects in the scene.

	Control			Gaze Guided		
	Subject 1	Subject 2	Subject 3	Subject 1	Subject 2	Subject 3
Completion Time (min:sec)	1:04	0:37	2:54	0:30	0:24	0:25
Errors	1	0	2	0	0	0

Figure 7.5: Performance of subjects on a parts retrieval task. Photograph shows initial arrangement of parts on the table.

Chapter 8

Conclusions

8.1 Conclusions & Future Work

The work presented extends the Subtle Gaze Direction technique to guide viewer attention in controlled real-world environments. By projecting subtle luminance modulations into the physical world, the system is capable of drawing attention to a target region very quickly (response time ≈ 0.5 seconds) and the process can be repeated on other parts of the scene to guide the viewer attention over time. Results of a user study reveal that the chosen approach effectively guides viewers through sequences of objects with less than one error per sequence consisting of eight objects. Furthermore, the likelihood of more then one error is only 18%.

The main limitation encountered is restricted mobility due to the dependence on a fixed projector. It would be better to present the visual cues on a head-mounted display, however, commercial eye-trackers with head-mounted displays for augmented reality applications are not yet commonplace. Note that other researchers have attempted to track the eyes from below/above the frame of head-mounted displays with limited success. Transitioning to an integrated system would be simple and will greatly improve mobility and also be less distracting to bystanders.

There are algorithmic improvements that could be made to the system to improve reliability and performance. First, a significant limitation is the scene camera itself, which performs poorly in low-light environments and provides no information about camera exposure and white balancing to the API. Such information would be useful in developing a

case-by-case strategy for varying environmental conditions. Such an enhancement would allow for the replacement of the current state-based decision engine by a simple look-up table. Secondly, the gaze inference mechanism is susceptible to angular deflection from the horizon (user rolls his/her head). This can be mitigated by comparing the set of source keypoints to the set of matching scene keypoints to determine an approximate roll value. With the deflection established, apply an inverse roll operation to the set of vectors in the scene image to obtain a perspective consistent with the source image. All together, discover the approximate roll value, return the scene image to true horizon, and then perform the inference operation.

As for modifying the problem approach, an obvious next-step is to experiment with more complex and dynamic environments. Recently published work from the eye-tracking community which documents 'best practices' for collecting eye-tracking data in outdoor environments [48, 16] would be suitable follow-ups as well as work from computer vision and augmented-reality on object detection and tracking [66, 27].

Successfully directing an observer's gaze in the real world has many applications including:

- **Simulators:** A simulator could incorporate gaze manipulation directly into the display mechanism via a projector or multiple display panels. Using only classical SGD, the user would be restricted to a single display and narrow viewing angle. Here, the user is free to move their head side-to-side and up-and-down.

- **Augmented Reality:** With the incorporation of a head-mounted display, real-world gaze manipulation could be used to highlight virtually any object of interest without obstructing the user's view. The gaze directed item can be roughly any size and shape, and can be determined by the specific task the user wishes to accomplish. In theory, different software applications could dynamically change the functionality of real-world gaze manipulation. Such software applications could include:

- *Navigation:* GPS navigation could utilize real-world gaze manipulation to aid in visually locating destinations; such as a restaurant or social gathering.
- *Friend Finder:* Use gaze manipulation to direct a subject towards faces that match their friend's description.
- *Monument Finder:* Tourists could use an application to point out landmarks or milestones in their environment.
- *Self-guided Tour:* Exhibitors could develop an application to aid in user navigation through exhibits.
- *Study Aid:* If a subject strays from studying, real-world gaze manipulation can suggest they return to their notes.

In summary, this work explored the question of guiding a viewers' attention in a real world setting. The developed system extends the Subtle Gaze Direction paradigm to guide attention to physical objects and provides an exciting foundation for future work.

Bibliography

[1] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 510–517. IEEE, 2012.

[2] Jason S. Babcock and Jeff B. Pelz. Building a lightweight eyetracking headgear. In *Proceedings of the 2004 Symposium on Eye Tracking Research and Applications*, ETRA '04, pages 109–114, New York, NY, USA, 2004. ACM.

[3] Reynold Bailey, Ann McNamara, Aaron Costello, Srinivas Sridharan, and Cindy Grimm. Impact of subtle gaze direction on short-term spatial information recall. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12, pages 67–74, New York, NY, USA, 2012. ACM.

[4] Reynold Bailey, Ann McNamara, Nisha Sudarsanam, and Cindy Grimm. Subtle gaze direction. *ACM Trans. Graph.*, 28:100:1–100:14, September 2009.

[5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006*, pages 404–417. Springer, 2006.

[6] G. Bradski. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] James R Brockmole and John M Henderson. Recognition and attention guidance during contextual cueing in real-world scenes: evidence from eye movements. *Q J Exp Psychol (Hove)*, 59(7):1177–87, Jul 2006.

[8] K Chajka, M Hayhoe, B Sullivan, J Pelz, N Mennie, and J Droll. Predictive eye movements in squash. *Journal of Vision*, 6(6):481, 2006.

[9] J. Chastine, K. Nagel, Ying Zhu, and M. Hudachek-Buswell. Studies on the effectiveness of virtual pointers in collaborative augmented reality. In *Proceedings of the 2008 IEEE Symposium on 3D User Interfaces*, 3DUI '08, pages 117–124, Washington, DC, USA, 2008. IEEE Computer Society.

[10] Xin Chen and Gregory J. Zelinsky. Real-world visual search is dominated by top-down guidance. *Vision research*, 46(24):4118–4133, November 2006.

[11] R Dennis Cook. Detection of influential observation in linear regression. *Technometrics*, pages 15–18, 1977.

[12] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[13] D. DeCarlo and A. Santella. Stylization and abstraction of photographs. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 769–776, New York, NY, USA, 2002. ACM Press.

[14] Digia. Qt project. `http://qt-project.org/`.

[15] R. Dodge. Visual perception during eye movement. *Psychological Review*, 7:454–465, 1900.

[16] K. Evans, R. Jacobs, J. Tarduno, and J. Pelz. Collecting and analyzing mobile eye-tracking data in outdoor environments. *Journal of Eye Movement Research*, 5(2):6:1–19, 2012.

[17] Steffen Gauglitz, Cha Lee, Matthew Turk, and Tobias Höllerer. Integrating the physical environment into mobile remote collaboration. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services*, MobileHCI '12, pages 241–250, New York, NY, USA, 2012. ACM.

[18] Laurent Gomila. Simple and fast multimedia library. `http://www.sfml-dev.org/`.

[19] E.R. Grant and M. J. Spivey. Eye movements and problem solving: guiding attention guides thought. *Psychological Science*, 14(5):462–466, 2003.

[20] Martin Groen and Jan Noyes. Solving problems: How can guidance concerning task-relevancy be provided? *Comput. Hum. Behav.*, 26:1318–1326, November 2010.

[21] Pavel Gurevich, Joel Lanir, Benjamin Cohen, and Ran Stone. Teleadvisor: a versatile augmented reality tool for remote assistance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 619–622, New York, NY, USA, 2012. ACM.

[22] J. M. Henderson and A. Hollingworth. Eye movements during scene viewing: An overview. In G. Underwood, editor, *Eye Guidance in Reading and Scene Perception*, pages 269–293. Oxford: Elsevier., 1998.

[23] T. E. Hutchinson, K. P. White, W. N. Martin, K. C. Reichert, and L. A. Frey. Human-computer interaction using eye-gaze input. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(6):1527–1534, 1989.

[24] R. J. K. Jacob and K. S. Karn. *The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research*, chapter Eye Tracking in Human-Computer Interaction and Usability Research: Ready to Deliver the Promises (Section Commentary), pages 573–605. Elsevier Science, Amsterdam, 2003.

[25] John Jonides. *Voluntary versus automatic control over the mind?s eye's movement*, volume 9, pages 187–203. Erlbaum, 1981.

[26] Ievgen Khvedchenia. Comparison of the opencv's feature detection algorithms. `http://computer-vision-talks.com/2011/01/comparison-of-the-opencvs-feature-detection-algorithms-2/`.

[27] Georg Klein and Tom Drummond. Robust visual tracking for non-instrumented augmented reality. In *Proc. Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, pages 113–122, Tokyo, October 2003.

[28] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[29] V. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, 1:8–17, 1965.

[30] J. L. Levine. An eye-controlled computer. Research Report RC-8857, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., 1981.

[31] Rui Li, Jeff Pelz, Pengcheng Shi, Cecilia Ovesdotter Alm, and Anne R. Haake. Learning eye movement patterns for characterization of perceptual expertise. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12, pages 393–396, New York, NY, USA, 2012. ACM.

[32] Oskar Linde and Tony Lindeberg. Object recognition using composed receptive field histograms of higher dimensionality. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 1–6. IEEE, 2004.

[33] Oskar Linde and Tony Lindeberg. Composed complex-cue histograms: An investigation of the information content in receptive field based image descriptors for object recognition. *Computer Vision and Image Understanding*, 116(4):538–560, 2012.

[34] D. Litchfield and L. J. Ball. Using another's gaze as an explicit aid to insight problem solving. *Q J Exp Psychol (Hove)*, 64:649–656, Apr 2011.

[35] D. Litchfield, L. J. Ball, T. Donovan, D. J. Manning, and T. Crawford. Viewing another person's eye movements improves identification of pulmonary nodules in chest x-ray inspection. *J Exp Psychol Appl*, 16:251–262, Sep 2010.

[36] Margaret Livingstone and David H Hubel. *Vision and art: The biology of seeing*, volume 2. Harry N. Abrams New York, 2002.

[37] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[38] N. H. Mackworth and A. J. Morandi. The gaze selects informative details within pictures. *Perception and Psychophysics*, 2:547–552, 1967.

[39] S. K. Mannan, K. H. Ruddock, and D. S. Wooding. The relationship between the locations of spatial features and those of fixations made during visual examination of briefly presented images. *Spatial Vision*, 10:165–188, 1996.

[40] Ann McNamara, Reynold Bailey, and Cindy Grimm. Search task performance using subtle gaze direction with the presence of distractions. *ACM Trans. Appl. Percept.*, 6:17:1–17:19, September 2009.

[41] Microsoft. Windows forms. `http://msdn.microsoft.com/en-us/library/dd30h2yb(v=vs.110).aspx`.

[42] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 27(10):1615–1630, 2005.

[43] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP (1)*, pages 331–340, 2009.

[44] Marius Muja and David G Lowe. Fast matching of binary features. In *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, pages 404–410. IEEE, 2012.

[45] Evelyn C O'Neill, Yu Xiang George Kong, Paul P Connell, Dai Ni Ong, Sharon A Haymes, Michael A Coote, and Jonathan G Crowston. Gaze behavior among experts and trainees during optic disc examination: does how we look affect what we see? *Invest Ophthalmol Vis Sci*, 52(7):3976–83, Jun 2011.

[46] G. Osterberg. Topography of the layer of rods and cones in the human retina. Acta Ophthal., 1935. suppl. 6, 11-97.

[47] D. Parkhurst and E. Niebur. Scene content selected by active vision. *Spatial Vision*, 16:125–154, 2003.

[48] J. Pelz, T. Kinsman, and K. Evans. Analyzing complex gaze behavior in the natural world. *SPIE-IS&T Human Vision and Electronic Imaging XVI*, pages 1–11, 2011.

[49] Jeff Pelz, Roxanne Canosa, Diane Kucharczyk, Jason Babcock, Amy Silver, and Daisei Konno. Portable eyetracking: A study of natural eye movements. In *Human Vision and Electronic Imaging V, SPIE Proceedings*, page 3659, 2000.

[50] Ramesh Raskar, Kok Low, and Greg Welch. Shader lamps: Animating real objects with image-based illumination. Technical report, Chapel Hill, NC, USA, 2000.

[51] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006.

[52] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[53] Sajay Sadasivan, Joel S. Greenstein, Anand K. Gramopadhye, and Andrew T. Duchowski. Use of eye movements as feedforward training for a synthetic aircraft inspection task. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, pages 141–149, New York, NY, USA, 2005. ACM.

[54] Bernt Schiele and JamesL. Crowley. Recognition without correspondence using multidimensional receptive field histograms. *International Journal of Computer Vision*, 36(1):31–50, 2000.

[55] M. Sodhi, B. Reimer, J. L. Cohen, E. Vastenburg, R. Kaars, and S. Kirschenbaum. On-road driver eye movement tracking using head-mounted devices. In *Proceedings of the 2002 symposium on Eye tracking research and applications*, ETRA '02, pages 61–68, New York, NY, USA, 2002. ACM.

[56] Srinivas Sridharan, Reynold Bailey, Ann McNamara, and Cindy Grimm. Subtle gaze manipulation for improved mammography training. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12, pages 75–82, New York, NY, USA, 2012. ACM.

[57] Michael J. Swain and Dana H. Ballard. Color indexing. *Int. J. Comput. Vision*, 7(1):11–32, November 1991.

[58] Benjamin W. Tatler, Nicholas J. Wade, Hoi Kwan, John M. Findlay, and Boris M. Velichkovsky. Yarbus, eye movements, and vision. *i-Perception*, 1, 2010.

[59] L.E. Thomas and A. Lleras. Moving eyes and moving thought: on the spatial compatibility between eye movements and cognition. *Psychonomic bulletin and review*, 14(4):663–668, 2007.

[60] Antonio Torralba, Aude Oliva, Monica S. Castelhano, and John M. Henderson. Contextual guidance of eye movements and attention in real-world scenes: the role of global features in object search. *Psychological review*, 113(4):766–786, October 2006.

[61] Geoffrey Underwood and Tom Foulsham. Visual saliency and semantic incongruency influence eye movements when inspecting pictures. *Q J Exp Psychol (Hove)*, 59(11):1931–49, Nov 2006.

[62] P. Vaidyanathan, J. Pelz, Rui Li, S. Mulpuru, Dong Wang, Pengcheng Shi, C. Calvelli, and A. Haake. Using human experts' gaze data to evaluate image processing algorithms. In *IVMSP Workshop, 2011 IEEE 10th*, pages 129 –134, june 2011.

[63] Eduardo E. Veas, Erick Mendez, Steven K. Feiner, and Dieter Schmalstieg. Directing attention and influencing memory with visual saliency modulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1471–1480, New York, NY, USA, 2011. ACM.

[64] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.

[65] MéLodie Vidal, Jayson Turner, Andreas Bulling, and Hans Gellersen. Wearable eye tracking for mental health monitoring. *Comput. Commun.*, 35(11):1306–1311, June 2012.

[66] Daniel Wagner, Dieter Schmalstieg, and Horst Bischof. Multiple target detection and tracking with guaranteed framerates on mobile phones. In *Proceedings of the 2009 8th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '09, pages 57–64, Washington, DC, USA, 2009. IEEE Computer Society.

[67] Dirk Walther, Ueli Rutishauser, Christof Koch, and Pietro Perona. Selective visual attention enables learning and recognition of multiple objects in cluttered scenes. *Comput. Vis. Image Underst.*, 100(1-2):41–63, October 2005.

[68] Ranxiao Frances Wang and Elizabeth S. Spelke. Human spatial representation: insights from animals. *Trends in Cognitive Sciences*, 6(9):376 – 382, 2002.

[69] P. Wellner and S. Freeman. The doubledigitaldesk: Shared editing of paper documents. Technical Report Tech. Rep. EPC-93-108, Xerox EuroPARC, 1993.

[70] A. L. Yarbus. *Eye Movements and Vision*. Plenum. New York., 1967.

Appendix A

Runtime Testing

This Appendix lists the testing procedure utilized in Section 5.1. The testing program includes a means of selecting the keypoint detection and descriptor extraction algorithms at runtime via command-line arguments. GPU-acceleration and radial matching may be enabled through preprocessor definitions. All testing parameters related to detection and extraction, except when explicitly mentioned in Section 5.1, are the API defaults. For radial distance matching a distance of 100 was used. That value was chosen based on the understanding of distance limiting established in Section 5.1.3.

test.cpp:

```
#include <stdio.h>
#include <iostream>
#include <Windows.h>
#include <opencv2\core\core.hpp>
#include <opencv2\features2d\features2d.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\calib3d\calib3d.hpp>
#include <opencv2\nonfree\nonfree.hpp>
#include <opencv2\gpu\gpu.hpp>

// #define GPU_MATCHING for GPU-acceleration
#ifndef GPU_MATCHING
#define CPU_MATCHING
#endif
// #define RADIAL_MATCH for non-global matching
#ifndef RADIAL_MATCH
#define GLOBAL_MATCH
#endif

using namespace cv;

// These strings define the supported detectors / extractors
// for this test. Factory methods in OpenCV use these strings
// to construct the correct algorithm implementation. The 3rd
// and 4th input arguments must match one item from both sets.
```

```
26 string detectors_strs [] = {
27     "FAST",
28     "STAR",
29     "SIFT",
30     "SURF",
31     "ORB",
32     "BRISK",
33     "MSER"
34 };
35
36 string extractor_strs [] = {
37     "SIFT",
38     "SURF",
39     "BRIEF",
40     "BRISK",
41     "ORB",
42     "FREAK"
43 };
44
45 // Usage: test IMG1 IMG2 DETECTOR EXTRACTOR
46 int main( int argc, char** argv )
47 {
48   // load scene and source
49   Mat img_object = imread( argv[1], CV_LOAD_IMAGE_GRAYSCALE );
50   Mat img_scene = imread( argv[2], CV_LOAD_IMAGE_GRAYSCALE );
51   Ptr<FeatureDetector> detector = FeatureDetector::create(argv[3]);
52   Ptr<DescriptorExtractor> extractor = DescriptorExtractor::create(argv[4]);
53
54   // image resources
55   std::vector<KeyPoint> keypoints_object, keypoints_scene;
56   Mat descriptors_object, descriptors_scene;
57   // performance metrics
58   LARGE_INTEGER freq, start, finish;
59   QueryPerformanceFrequency(&freq);
60
61   //
62   // STAGE 1: Detect keypoints
63   //
64   detector->detect( img_object, keypoints_object );
65
66   QueryPerformanceCounter(&start);
67   detector->detect( img_scene, keypoints_scene );
68
69   QueryPerformanceCounter(&finish);
70
71   std::cout << "Detection time: " <<
72         (double(finish.QuadPart - start.QuadPart)/freq.QuadPart) << std::endl;
73   std::cout << "Number of keypoints: " << keypoints_scene.size() << std::endl;
74
75   //
76   // STAGE 2: Extract Descriptors
77   //
78   extractor->compute( img_object, keypoints_object, descriptors_object );
79     QueryPerformanceCounter(&start);
80   extractor->compute( img_scene, keypoints_scene, descriptors_scene );
81   descriptors_object.convertTo(descriptors_object, CV_32F);
82   descriptors_scene.convertTo(descriptors_scene, CV_32F);
83
```

```
84    QueryPerformanceCounter(&finish);
85
86    std::cout << "Extraction time: " <<
87          (double(finish.QuadPart - start.QuadPart)/freq.QuadPart) << std::endl;
88
89    //
90    // STAGE 3: Descriptor Matching
91    //
92  #ifdef RADIAL_MATCH
93    vector<vector< DMatch >> matches;
94  #else
95    vector<DMatch> matches;
96  #endif
97  #ifdef GPU_MATCHING
98    // upload object descriptors
99    gpu::GpuMat objDesc_gpu( descriptors_object );
100   gpu::BruteForceMatcher_GPU_base matcher_gpu;
101   // Each frame will need to be uploaded to the GPU. Therefore, we will
102   // include the upload time in the computation.
103   QueryPerformanceCounter(&start);
104   gpu::GpuMat scnDesc_gpu( descriptors_scene );
105 #ifdef RADIAL_MATCH
106   matcher_gpu.radiusMatch(objDesc_gpu, matches, 100.0f);
107 #else
108    matcher_gpu.match(objDesc_gpu, scnDesc_gpu, matches);
109 #endif
110 #else
111 // CPU-based Matching (Optionally Radial)
112 FlannBasedMatcher matcher;
113 QueryPerformanceCounter(&start);
114 #ifdef RADIAL_MATCH
115   matcher.radiusMatch( descriptors_object, descriptors_scene, matches, 100.0f);
116 #else
117   matcher.match( descriptors_object, descriptors_scene, matches);
118 #endif
119 #endif
120    QueryPerformanceCounter(&finish);
121    std::cout << "Matching time: " <<
122        (double(finish.QuadPart - start.QuadPart)/freq.QuadPart) << std::endl;
123
124   Mat img_matches;
125   drawMatches(  img_object, keypoints_object,
126                 img_scene, keypoints_scene,
127                 matches, img_matches,
128                 Scalar::all(-1), Scalar::all(-1),
129 #ifdef RADIAL_MATCH
130                 vector<vector<char>>(), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
131 #else
132                 vector<char>(), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
133 #endif
134
135   //-- Show detected matches
136   imshow( "Good Matches & Object detection", img_matches );
137
138   waitKey(0);
139
140   return 0;
141 }
```

Appendix B

Gaze Inference Derivation

Beginning with the elementary equation for point-slope:

$$(y_1 - y_2) = m(x_1 - x_2) \tag{B.1}$$

Where m is defined as the Cartesian slope between unique points $\{x_1, y_1\}$ and $\{x_2, y_2\}$. Defining the source descriptors as$\{x_a, y_a\}$, the scene descriptors as $\{x_b, y_b\}$, the scene gaze point as $\{e_x, e_y\}$, and the inferred gaze as $\{e'_x, e'_y\}$, yields two equations that relate scene and source via m:

$$(y_b - e_y) = m(x_b - e_x) \tag{B.2}$$

$$(y_a - e'_y) = m(x_a - e'_x) \tag{B.3}$$

Composing both equations together:

$$(y_a - e'_y) = \frac{(y_b - e_y)}{(x_b - e_x)}(x_a - e'_x)$$

Then distributing by $(x_b - e_x)$:

$$(y_a - e'_y)(x_b - e_x) = (y_b - e_y)(x_a - e'_x)$$

Expanding:

$$y_a x_b - y_a e_x - e'_y x_b + e'_y e_x = y_b x_a - y_b e'_x - e_y x_a + e_y e'_x$$

Separating $\{e'_x, e'_y\}$ and $\{e_x, e_y\}$:

$$e'_y e_x - e'_y x_b + y_b e'_x - e_y e'_x = y_a e_x - y_a x_b + y_b x_a - e_y x_a$$

Factor by $\{e'_x, e'_y\}$:

$$(e_x - x_b)e'_y + (y_b - e_y)e'_x = y_a e_x - y_a x_b + y_b x_a - e_y x_a$$

Express as an over-constrained system of equations:

$$\begin{bmatrix} y_{b1} - e_y & e_x - x_{b1} \\ y_{b2} - e_y & e_x - x_{b2} \\ \vdots & \vdots \\ y_{bn} - e_y & e_x - x_{bn} \end{bmatrix} \begin{bmatrix} e'_x \\ e'_y \end{bmatrix} = \begin{bmatrix} y_{a1}e_x - y_{a1}x_{b1} + y_{b1}x_{a1} - e_y x_{a1} \\ y_{a2}e_x - y_{a2}x_{b2} + y_{b2}x_{a2} - e_y x_{a2} \\ \vdots \\ y_{an}e_x - y_{an}x_{bn} + y_{bn}x_{an} - e_y x_{an} \end{bmatrix} \tag{B.4}$$

Express the LHS as Ae' and the RHS as b:

$$Ae' = b$$

Solve the overdetermined system for e' (the inferred fixation point):

$$A^T A e' = A^T b$$

$$e' = (A^T A)^{-1} A^T b \tag{B.5}$$

Appendix C

MATLAB Prototype

cameraSetup.m:

```
function cameraSetup
%CAMERASETUP Webcam setup helper. Creates global "camObj"
% using IMAQ package.
clear all;
clc;
imaqreset;
assignin('base', 'camObj', imaq.VideoDevice)

end
```

cockpitdemo.m:

```
% Fixation matching demonstration using SIFT (static image)

% Testing clears
clear all;
clc;

figure(1); clf;

% ------------------------------------------------------------------------
%                                                              Thresholds
% ------------------------------------------------------------------------

% SIFT-related
% Increase peakthresh to return fewer features ( intensity )
PeakThresh = 0.037;
% Decrease (non)edgethresh to return more features ( quantity )
EdgeThresh = 0.0;

% Fixation Matching
% Increase Cook Thresh allow more influential ( distant matches )
CookThresh = 0.6; % works in well-constrained cases

testno = 4; %1 - 6
%eye_pos = [ 495, 220 ];
eye_pos = [450, 220];

% scanning radius and associated boundaries
scan_radius = 100;

```

```
30 % ------------------------------------------------------------------------
31 %                                                          Create image pair
32 % ------------------------------------------------------------------------
33
34 CacheDir = 'cache';
35 if(exist(CacheDir, 'dir') == 0)
36     mkdir(CacheDir);
37 end;
38
39 ImageDir = 'images';
40 SourceDir = 'sources';
41 sourceImg = 'cockpit_00';        % high-res source image
42
43 load(fullfile(ImageDir, 'imgList.mat'));
44 sceneImg = strtrim(testImgs(testno,:));  % webcam or "scene" camera
45 ext = '.jpg';
46
47 % Source image
48 Ia = imread(fullfile(ImageDir, SourceDir, [sourceImg ext]));
49 % scene image
50 Ib = imread(fullfile(ImageDir, [sceneImg ext]));
51 Ib = imresize(Ib, 1.0);
52 % resize image
53 Ia = im2window(Ia, [size(Ib,1), size(Ib,2)] );
54
55 % ------------------------------------------------------------------------
56 %                                          Extract features and match
57 % ------------------------------------------------------------------------
58
59 filecache = fullfile(CacheDir, [sourceImg '.mat']);
60 if(exist(filecache, 'file'))
61     fprintf('SIFT cache found for source image: %s\n', sourceImg);
62     load(filecache);
63 else
64     fprintf('Creating SIFT features for source image: %s\n', sourceImg);
65     [fa,da] = vl_sift(im2single(rgb2gray(Ia)), ...
66                       'PeakThresh',   0.035,    ...
67                       'EdgeThresh',   15, ...
68                       'Octaves', 1000) ;
69     save(filecache, 'fa', 'da');
70 end;
71
72 tic;
73 [fb,db] = vl_sift(im2single(rgb2gray(Ib)), ...
74           'PeakThresh',   0.0,    ...
75           'EdgeThresh',   30) ;
76 toc;
77 tic;
78 matches = vl_ubcmatch(da,db,2.0);
79 toc;
80
81 % figure(1) ; clf ;
82 % imagesc(cat(2, Ia, Ib)) ;
83 %  axis image off ;
84
85 figure(1) ; clf ;
86 imagesc(cat(2, Ia, Ib)) ;
87
```

```
% eye position simulation
marker_size = 16;
marker_dim(1:2) = marker_size;
marker_pos = [ eye_pos(1) + size(Ia,2) - marker_size/2 ...
    eye_pos(2) - marker_size/2 ];
rectangle('Position', [ marker_pos, marker_dim], 'facecolor', 'r');

% TODO: if rectangle is outside image space, no match
x_max = size(Ib,2);
y_max = size(Ib,1);

x_start = eye_pos(1) - scan_radius;
x_end   = eye_pos(1) + scan_radius;

y_start = eye_pos(2) - scan_radius;
y_end   = eye_pos(2) + scan_radius;

% pre-allocation
selection = zeros(size(matches));
x_val = zeros(1);
y_val = zeros(1);
point = zeros(1);

% determine what matches lie in the scan radius
% TODO parallel (?)
for i=1:size(matches,2)
     % point in scene
     point = fb(:, matches(2,i));
     x_val = point(1);
     y_val = point(2);
     % that is constrained to the scan radius
     if( x_val >= x_start && x_val <= x_end && ...
             y_val >= y_start && y_val <= y_end)
         % select the point pair
         selection(:,i) = matches(:,i);
     end
end

% remove zero-valued elements
sel = find(selection(1,:));
selection = selection(:,sel);
hold on;

% plot source correspondence
vl_plotframe(fa(:, selection(1,:)), 'color', 'red');
% compute fb with only selections
fb_s = fb(:, selection(2,:));
% adjust for x
fb_s(1,:) = fb_s(1,:) + size(Ia,2);
% plot scene features
vl_plotframe(fb_s);

xa = fa(1,selection(1,:)) ;
xb = fb(1,selection(2,:)) + size(Ia,2) ;
ya = fa(2,selection(1,:)) ;
yb = fb(2,selection(2,:)) ;

hold on ;
```

```
146 h = line([xa ; xb], [ya ; yb]) ;
147 set(h,'linewidth', 1, 'color', 'b') ;
148
149 axis image off ;
150
151 % compute Cook's Distance for the match set and remove outliers
152 match_pts = [xa' ya'];
153 CookDist = cookdist(match_pts);
154 %outlier_ind = find(cook_dist > 5/numel(xa));
155 outlier_ind = find(CookDist > 8.6572*numel(CookDist)^(-1.284));
156 xa(outlier_ind) = [];
157 xb(outlier_ind) = [];
158 ya(outlier_ind) = [];
159 yb(outlier_ind) = [];
160
161 % compute radius and angle from fixation and (non-outlier) feature points at the
162 % scene
163 xb =  xb - size(Ia,2);
164
165 ex = eye_pos(1);
166 ey = eye_pos(2);
167
168 A = zeros(numel(xa), 2);
169 A(:,1) = (yb - ey)';
170 A(:,2) = (ex - xb)';
171 b = (xa.*yb - xb.*ya + ya.*ex - xa.*ey)';
172
173 % A'Ax = A'b
174 xls = (A'*A)\(A'*b);
175
176 % if ( pts lie on one side)
177 % find point with least LSE
178 % linreg = polyfit(
179
180 % get distance from point and inferred fixation
181 % reverse direction of radial direction and replot fixation point
182
183 % Inferred eye position
184 eye_inf = [ (xls(1) - marker_size/2) (xls(2) - marker_size/2) ];
185
186 % draw inference paths
187 xa(2, :) = xls(1);
188 ya(2, :) = xls(2);
189 line(xa, ya)
190
191 rectangle('Position', [ eye_inf, marker_dim], 'facecolor', 'g');
192 if( eye_inf(1) < 0 || eye_inf(2) < 0 || ...
193         eye_inf(1) > size(Ia,2) || eye_inf(2) > size(Ia,1) )
194     warning('Inferred fixation exceeds image bounds!'); %#ok<WNTAG>
195 end
```

cookdist.m:

```
1 function [ dist ] = cookdist( D )
2 %COOKDIST Cook's distance function
3 %
4 % m = mean(X);
5 %
```

```
% e = mean((m - X).^2);

[n,c] = size(D);

Y = D(:,c);                  %response vector
X = [ones(n,1) D(:,1:c-1)]; %design matrix
p = size(X,2);               %number of parameters

% Least Squares Estimation
b = (X'*X)\(X'*Y);
Ye = X*b;                    %expected response value
e = Y-Ye;                    %residual term
SSRes = e'*e;                %residual sum of squares
rb = size(b,1);
v2 = n-rb;                   %residual degrees of freedom
MSRes = SSRes/v2;            %residual mean square
Rse = sqrt(MSRes);           %standard error term
H = X*((X'*X)\X');
hii = diag(H);               %leverage of the i-th observation
ri = e./(Rse*sqrt(1-hii));  %residual

%Cook's distance
dist = diag((ri.^2/p)*((hii./(1-hii)))');

end
```

im2window.m:

```
function [ im_out ] = im2window( im_in, out_dims )
%IM2WINDOW Clamps image to window frame and pad to match dimensions
%   The ubc_match function in vl_feat requires both images to have the same
%   dimensions. It considers padding the image (with black)
%   should the image not resize exactly to the new dimensions.

% set dims
in_h = size(im_in, 1);
in_w = size(im_in, 2);
out_h = out_dims(1);
out_w = out_dims(2);

in_ratio = in_w/in_h;
out_ratio = out_w/out_h;

% image sizes are identical
if(eq(in_w, out_w) && eq(in_h, out_h))
    im_out = im_in;
% input image is wider
elseif(in_ratio > out_ratio)
    % resize image
    im_temp = imresize(im_in, out_w/in_w);
    % pad with black
    resize_h = size(im_temp,1);
    pad_h_t = floor((out_h - resize_h)/2);
    im_out = zeros(out_h, out_w, 3, 'uint8');
    im_out(pad_h_t:pad_h_t+resize_h -1,:,:) = im_temp(:,:,:);
% input image is taller
elseif(in_ratio < out_ratio)
    % resize image
```

```
31      im_temp = imresize(im_in, out_h/in_h);
32      % pad with black
33      resize_w = size(im_temp,2);
34      pad_w_l = ceil((out_w - resize_w)/2);
35      im_out = ones(out_h, out_w, 3, 'uint8');
36      im_out(:, pad_w_l:pad_w_l+resize_w-1,:) = im_temp;
37  % input image has identical dimensions, but different size
38  else
39      % input image is larger
40      if(in_w > out_w)
41          im_out = imresize(out_w/in_w);
42      % input image is smaller
43      elseif(in_w < out_w)
44          im_out = imresize(out_w/in_w);
45      else
46          error('Logical Error: Image is not equal, and not scalable');
47      end
48  end
49
50  % case handled gracefully
51  return;
```

eyeMatch.m:

```
1   function eyeMatch( srcImg, camObj )
2   %EYEMATCH Imposes a sample fixation on a designated source image. Incoming
3   %frames are provided via a webcam specified by "camObj"
4
5       eyePos = [450, 200];
6
7   % ------------------------------------------------------------------------
8   %                                                              Thresholds
9   % ------------------------------------------------------------------------
10
11      % SIFT-related
12      % Increase peakthresh to return fewer features ( intensity )
13      PeakThresh = 0.037;
14      % Decrease (non)edgethresh to return more features ( quantity )
15      EdgeThresh = 0.0;
16      % How "hard" the algorithm works to produce a match
17      matchThresh = 2.0;
18
19      % Fixation Matching
20      % Increase Cook Thresh allow more influential ( distant matches )
21      CookThresh = 0.6; % works in well-constrained cases
22      % scanning radius and associated boundaries
23      scan_radius = 100;
24      % Marker Size
25      markerSize = 16;
26      % Marker Dimensions
27      markerDim = [markerSize markerSize];
28
29      % Fixation graphics handles
30      eyeFix_h = 0;
31      eyeInf_h = 0;
32
33  % ------------------------------------------------------------------------
34  %                                                            Source SIFT
```

```
35 %--------------------------------------------------------------------------
36
37     % Target Directories
38     ImageDir = 'images';
39     SourceDir = 'sources';
40     ext = '.jpg';
41
42     % pre-allocation
43     x_val = zeros(1); %#ok<*NASGU>
44     y_val = zeros(1);
45     point = zeros(1,2);
46     input_h = figure( 'CloseRequestFcn', @closeCB);
47     output_h = figure( 'CloseRequestFcn', @closeCB);
48
49     % Pre-compute SIFT for source
50     Ia = imread(fullfile(ImageDir, SourceDir, [srcImg ext]));
51     Ia = im2window(Ia,  [camObj.ROI(4) camObj.ROI(3)] );
52
53     [r c d] = size(Ia);
54     Ia_g = rgb2gray(Ia)';
55     Ia_g = reshape(Ia_g, [r c]);
56
57     [fa,da] = vl_sift(im2single(rgb2gray(Ia)), ...
58                       'PeakThresh',   0.035,  ...
59                       'EdgeThresh',   15) ;
60     %[da fa] = yasift(Ia_g);
61
62     set(0, 'CurrentFigure', output_h);
63     imshow(Ia);
64
65     wtrue = true;
66     while( wtrue )
67         % capture frame
68         Ib = step(camObj);
69         [r c d] = size(Ib);
70         Ib_g = rgb2gray(Ib)';
71         Ib_g = reshape(Ib_g, [r c]);
72         set(0, 'CurrentFigure', input_h);
73         imshow(Ib);
74         rectangle('Position', [ eyePos- markerSize/2, markerDim], 'facecolor', 'g');
75         drawnow;
76
77         % SIFT match (TODO: speedup?)
78         [fb,db] = vl_sift(im2single(rgb2gray(Ib)), ...
79                   'PeakThresh',   0.0,    ...
80                   'EdgeThresh',   30) ;
81         %[db fb] = yasift(Ib_g);
82         matches = vl_ubcmatch(da,db,matchThresh);
83
84         % compute scan radius bounds (technically a box)
85         x_start = eyePos(1) - scan_radius;
86         x_end   = eyePos(1) + scan_radius;
87
88         y_start = eyePos(2) - scan_radius;
89         y_end   = eyePos(2) + scan_radius;
90
91         % pre-allocate selections
92         radialMatches = zeros(size(matches));
```

```
93
94          % determine what matches lie in the scan radius
95          % TODO parallel (?) Seems slower
96          for i=1:size(matches,2)
97              % point in scene
98              point = fb(:, matches(2,i));
99              x_val = point(1);
100             y_val = point(2);
101             % that is constrained to the scan radius
102             if( x_val >= x_start && x_val <= x_end && ...
103                     y_val >= y_start && y_val <= y_end)
104                 % select the point pair
105                 radialMatches(:,i) = matches(:,i);
106             end
107         end
108
109         % clear zeros
110         selection = radialMatches(:, radialMatches(1,:) ~= 0 );
111
112         % no matches (need at least two pairs [4 elts])
113         if(numel(selection) <  4)
114             continue;
115         end;
116
117         xa = fa(1,selection(1,:));
118         xb = fb(1,selection(2,:));
119         ya = fa(2,selection(1,:));
120         yb = fb(2,selection(2,:));
121
122         % compute Cook's Distance for the match set and remove outliers
123         % The exponential components are the results of best-fit analysis
124         % accross multiple test cases.
125         match_pts = [xa' ya'];
126         CookDist = cookdist(match_pts);
127         outlier_ind = find(CookDist > 8.6572*numel(CookDist)^(-1.284));
128         xa(outlier_ind) = [];
129         xb(outlier_ind) = [];
130         ya(outlier_ind) = [];
131         yb(outlier_ind) = [];
132
133         % compute radius and angle from fixation and (non-outlier) feature
134         % points at the scene
135         ex = eyePos(1);
136         ey = eyePos(2);
137
138         A = zeros(numel(xa), 2);
139         A(:,1) = (yb - ey)';
140         A(:,2) = (ex - xb)';
141         b = (xa.*yb - xb.*ya + ya.*ex - xa.*ey)';
142
143         % A'Ax = A'b
144         xls = (A'*A)\(A'*b);
145
146         % When the algorithm diverges, the match is incorrect
147         % This will happen when the system is poorly constrained
148         % with either incorrect or garbage reference points
149         if(isfinite(xls))
150
```

```
151             % Inferred eye position (from Least-Squares)
152             eyeInf = [ (xls(1) - markerSize/2) (xls(2) - markerSize/2) ];
153
154             % delete graphics object if it exists
155             set(0, 'CurrentFigure', output_h);
156             if(eyeInf_h)
157                 delete(eyeInf_h);
158                 eyeInf_h = 0;
159             end
160             % set inferred eye position
161             eyeInf_h = rectangle('Position', [ eyeInf, markerDim], 'facecolor', 'g');
162             if( eyeInf(1) < 0 || eyeInf(2) < 0 || ...
163                     eyeInf(1) > size(Ia,2) || eyeInf(2) > size(Ia,1) )
164                 warning('Inferred fixation exceeds image bounds!');  %#ok<WNTAG>
165             end
166             % flush
167             drawnow;
168         end
169     end % Scaning Loop
170
171     % Clean-up
172     close force all;
173     release(camObj);
174
175     function closeCB( src, evt ) %#ok<INUSD>
176     %CLOSECB This function replaces CLOSEREQ as the callback for
177     % CLOSEREQFCN
178     %   Detailed explanation goes here
179         wtrue = false;
180     end
181 end % eyeMatch
```