Design of a Saccadic Active Vision System

by

Winnie Wong

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2006

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Human vision is remarkable. Through clever physical and neurological mechanisms, we are able to extract useful information from the light reflected by objects around us. Generations of evolution has fine-tuned these mechanisms for their tasks; an example of an optimized eye function is foveated vision. By limiting the main concentration of high-acuity photoreceptors to a small central region in the eye, we efficiently view the world by redirecting the fovea from point-of-interest to point-of-interest using eye movements called ***saccades***. This results in a very effective and resource-efficient visual acquisition system that can quickly respond and adapt to the real world.

Active vision is a special branch of computer vision where processing of visual data is reserved for relevant regions in the scene. Typically, active vision applications require real-time analysis of scene information; it is therefore necessary to quickly filter out data that does not contribute to the system's accomplishment of its task.

This thesis comprises two main parts. Part I describes a saccadic vision system that uses a commercial, programmable image sensor mounted on a servomotor. The dual-resolution saccadic camera is able to detect objects of interest in a scene based on processing of low-resolution image information, and then revisit the salient regions of the same scene in high-resolution. The location of foveal fixations is determined through an assignment of salience to each coordinate of the low-resolution image, where salience is determined by the presence of various stimuli in the local neighbourhood. The end product is a dual-resolution image in which background information is displayed in low-resolution, and salient areas are captured in high-acuity and locally appropriate exposures. This lends to a resource-efficient active vision system that can be used in numerous computer vision and control applications.

Part II describes CMOS image sensor design considerations for active vision applications. Specifically, this discussion focuses on methods to determine regions of interest and achieve high dynamic range on the sensor.

Acknowledgements

My time at York University and my return to University of Waterloo resulted in some truly rewarding, albeit unexpected, experiences. I would like to thank each and every person who made these past few years so enjoyable and so enriching. I can honestly say (without shame or banality), that thanks to you all, my dreams are coming true.

This feels like an Oscar moment.

To my father, the electrical engineer, who paved the way. To my mother, the devoted mother, who taught me the meaning of bravery and dignity. Thank you both, for all your sacrifices and support.

To my dragonboat team, The Aquaholics, for beating Harvard on their own river, and for reminding me that there is life beyond school and stress.

To the staff and participants at GoodLife and Tait Mackenzie, who helped me understand that the body is as important as the mind.

To Mr. Carr for allowing us the use of his cottage/palace, and to Mr. Kang, for inviting us to speak in China. Without you, I would have been stuck testing images of the lab and Lena.

To Yu Bai, Zhen Yang, Jilan Yang, and Elham Ashari for making life at Waterloo fun and interesting.

To Flora Li, Jackson Lai and Jeff Chang, for teaching me about photodiodes and delivering work reports for me.

To Ji-Soo Lee and Rajashekar Venkatachalam for their *gentle nudging* to apply for grad school.

To Yun Chan, for being there when I needed a friend most and for introducing me to the VISOR group.

To Lawrence Soung-Yee for his excellent advice on career moves and Belgian beer.

To Henok Mebrahtu for teaching me how to use the jebana and finding ways to dumb down difficult concepts for me. To Pete Carr for teaching me to waterski and educating me on hockey. To Wei Gao for helping to dilute the testosterone levels in the lab and critiquing my thesis (xie xie jie jie). To Rubakumar Krishnasamy for putting up with my bad moods. To Srdjan Pepic for

taking nothing seriously. To Eugene Savchenko for being the coolest and most grounded young genius ever. And to Anthony Badali for being a good sport and making me laugh.

To the computer science technical team at York University for saving my butt during every emergency.

To the Canadian Microelectronics Corporation for the use of their design tools and fabrication facilities, and to NSERC Canada and CRESTech for their funding.

To Dr. Paul Thomas for challenging my every word and keeping me honest.

To Dr. Manoj Sachdev and Dr. Mohab Anis for evaluating my thesis.

To my dedicated proofreaders, Betty Luk and Sylvia Wong, who put up with all my last-minute emails and ranting.

To Xiuling Wang for sharing her ideas with me.

To Benjamin Boomhour, who witnessed the very worst of me and survived.

To Edward Shen, who bailed me out of copious close-calls, and, in spite of my sass, granted me enormous amounts of patience.

Finally, I would like to express my deepest gratitude towards my teacher, supervisor, and friend, Dr. Richard Hornsey, who took pity on me when I was fresh and green, introduced me to the world of image sensors, and opened doors for me that I never knew existed. Thank you very much Richard, you will always be *The Best Boss Ever*!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> *"It might be said that by moving from the centre of the human retina to its periphery we travel back in evolutionary time; from the most highly organized structure to a primitive eye, which does little more than detect movements of shadows. The very edge of the human retina... gives primitive unconscious vision; and directs the highly developed foveal region to where it is likely to be needed for its high acuity."*
>
> — R.L. Gregory [1]

Look up for a moment from this thesis. What do you see? Chances are, without moving your head, you can see almost 180° of the scene before you. If you were to pay attention to the objects in the room, you would perhaps note the various pieces of furniture or assess the facial expressions of the people in your company. Furthermore, assuming that all the major objects are within 20 feet, you would likely agree that you can see most of the 180° view in front of you with equal clarity.

Intuition tells us that our eyes are like real-time, high resolution cameras that can see as clearly as the camera lens will allow. But intuition does not take into account the enormity of resources and processing that would be required to provide an immediate response to abundant visual details. Even with the brain's extraordinary complexity, how can it possibly process 180° worth of high-acuity visual information *instantaneously*? It turns out that, while human vision is indeed real-time, the eye is not a constantly high-resolution device. In fact, the human eye is

anisotropic: its acuity decreases eccentrically from the centre. The high-resolution fovea, which is responsible for high-acuity vision, is essentially time-shared; it jumps from point-of-interest to point-of-interest across the visual field. Our brain then reconstructs a coherent interpretation of the scene based on a combination of foveal details and generalized low-acuity information acquired from the eye's periphery. The eye is therefore a highly resource-efficient data-acquisition instrument that has been optimized through generations of evolution to reduce the amount of information that must be transmitted through the optic nerve and processed by the brain. Now how can this naturally optimized elegance be applied towards an engineering problem?

1.1 Research Motivation

This research project was inspired by the recorded eye movement patterns shown in Figure 1.1. The subject spends most of the time and attention on the eyes, nose and mouth, and performs a rough examination of the outline of the hair. A large percentage of the face (i.e. cheeks) does not receive attention from the fovea.

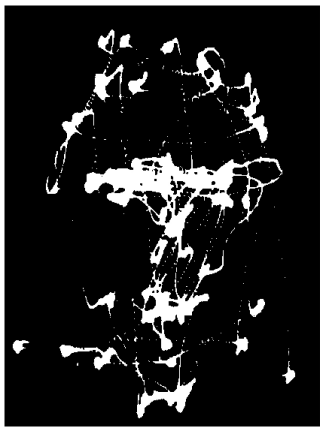


Figure 1.1: Visual examination of a face. Each dot represents a time unit spent on that location [2].

Yarbus is generally credited with providing the first modern understanding of human eye movements. He noted that:

> *"When looking at a human face, an observer usually pays most attention to the eyes, the lips, and the nose. The other parts of the face are given much more cursory consideration. The human eyes and lips are the most mobile and expressive elements of the face. The eyes and lips can tell an observer the mood of a person and his attitude towards the observer... It is therefore absolutely natural and understandable that the eyes and lips attract the attention more than any other part of the human face [3]."*

The question, however, is whether the eyes, nose and lips of a human face represent instinctively stimulating features to which an observer's attention is naturally drawn, or whether an observer's interest in these elements is a conditioned response upon which we are trained through experience to study. That is, would an untrained machine, designed to look for raw stimulating features in a general scene, fixate on the same parts of a face to which a human observer would attend?

The ability to use low resolution information to infer the contents of a scene has many interesting consequences. For example, it allows several different people to look at an abstract work of art and agree on the interpretation. This is the basis of the Impressionism, a painting style where the context of the artwork is implied through broad brushstrokes; for example, see Figure 1.2. These brushstrokes make little sense when considered on their own, and seen up close, an Impressionist painting would appear to be a low resolution 'blob'. However, when viewed from a distance, the brain is able to make sense of the combination of brushstrokes and form an understanding of the context. This implies that we really only need to see most of a scene in low resolution, and only certain important areas need to contain high resolution detail. This further means that valuable resources are not wasted in processing an abundance of detail that may not necessarily contribute to a better understanding of the scene. The resources of our visual system are therefore used efficiently.

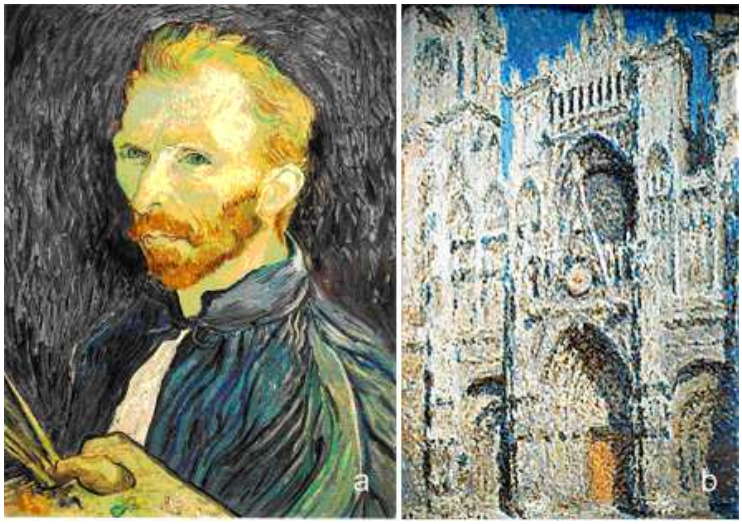


Figure 1.2: a) Self portrait of Vincent van Gogh [4]. Most of the information is inferred through broad brushstrokes. However, the most important part of the painting, the expressive eyes, contains finer detail. b) Rouen Cathedral by Claude Monet [5]. The details of the cathedral facade are implied through the combination of coarse brushstrokes.

1.2 Key Engineering Benefits

An engineering design based on the principle that the context of a scene can be understood by roughly analyzing a low-resolution view and attending only to the salient points of interest has several engineering benefits, including reduced processing and reduced transmission bandwidth requirements, allowing for a real-time response to the visual field. Such a system would be appropriate for applications such as the navigation of autonomous vehicles, object recognition, target tracking, robotic control, security monitoring, and industrial inspection (quality control). These applications require some method by which excess and irrelevant information can be filtered. Although data compression, such as runlength encoding[1] employed by JPEG (Joint Photographic Experts Group), can be used to alleviate transmission traffic, it does not address the problem of

[1]Runlength encoding is a lossless scheme, where redundant strings are reduced to a shorter description. For example:
Uncompressed data: ABCCCCCCCCDEFGGG, Compressed data: ABC!8DEFGGG [6]
The '!' is used as an M-byte flag to mark the beginning of encoded data. The 8 following the M-byte indicates the amount of redundancy.

processing the complex image.

Active vision is a branch of computer vision where processing is reserved for selected regions of the image. Typical active vision applications require specific, real-time information for control purposes, such as for robotic or autonomous vehicle guidance. Such projects employ various artificial intelligence techniques to determine how to efficiently and effectively allocate system resources in order to quickly extract information.

1.3 An Engineering Problem

In the summer of 2003, NASA launched the two Mars Exploration Rovers, Spirit and Opportunity, in hopes of gaining insight on the inhospitable planet's geology. To investigate the planetary surface, the Rovers traverse the Mars terrain outfitted with visual navigation systems that provide position and heading estimations to help guide their paths [7]. There are several potential issues to consider in the design of an autonomous navigation system, including:

- real-time processing of visual data (in response to changes in the terrain);
- tradeoffs between resolution, processing time and power, and data transmission bottlenecks;
- field of view; and
- fault tolerance and adaptability to the environment.

The autonomous navigation of the Mars Rover is just one example of an application that requires a real-time, resource efficient, and adaptable visual acquisition methodology. Other applications include industrial inspection, security surveillance, and target tracking.

1.4 A Proposed Solution

Project Mike, named for a fictional Cyclops-like character [8], is an active vision system that borrows from the design of its biological inspiration: the human vision system. Like the eye, Mike has a small high resolution fovea, which selectively attends to salient regions of a scene. It is an electronically and mechanically reconfigurable "saccadic" camera system, that efficiently examines

scenes through foveated imaging, where scrutiny is reserved for salient regions of interest. The system's "eye" is an electronic image sensor used in dual modes of resolution. A subwindow set at high resolution acts as the system's fovea, and the remaining wide-angle visual field is captured at a lower resolution. The ability to program the subwindow's size and position provides an analog to biological eye movements. Similarly, the system's mechanical components can be programmed to provide the "neck's" locomotion for modified perspectives.

Mike is an active vision design that can be used for industrial inspection, security, and autonomous guidance systems, such as the system required by the Mars Rovers.

1.5 Related Projects

Resulting from generations of evolution, designs in nature are often optimized for their functions and their environment. The human vision system is a particularly impressive information acquisition system, and lessons can be drawn from its design. Resources are efficiently utilized in foveated, selectively attentive vision: bulk processing is conducted on low-resolution information to extract regions of interest where further processing is required. This means that processing and data transmission resources are used economically.

There are numerous engineering applications for active vision systems, including autonomous navigation of vehicles, robotic guidance (visual servoing), and industrial inspection. Several works have reported projects on active vision systems that steer camera saccades mechanically [9], [10], [11], [12]. Some of these projects employ multiple cameras to achieve different resolutions for foveated imaging [12]. In 1999, the Jet Propulsion Laboratory reported the development of a non-mechanical, reconfigurable vision system with the ability to track targets using a multi-resolution sensor [13].

Visual servoing is a branch of computer vision closely related to active vision; it pertains to the use of cameras to guide robotic pose control. [14], [15], [16] describe the development of hand-eye coordinated robotic systems that rely on pre-calibrated vision (the eye) to calibrate robotic movements and tasks. In his robotic state estimation model, Bishop also considers the effect of lens distortion on the image [17]. It would be interesting to take the opposite approach, and use a known physical state to configure a vision system with uncharacterized image formation issues.

This will be further explored in Chapter 5.

1.6 Thesis Outline

This thesis comprises two main parts. Part I describes the Mike prototype, which uses a commercial, programmable image sensor mounted on a servomotor. The dual-resolution saccadic camera is able to detect objects of interest in a scene based on processing of low-resolution image information, and then revisit the salient regions of the same scene in high-resolution. The end product is a dual-resolution image in which background information is displayed in low-resolution, and salient regions are captured in high-acuity. This lends to a resource-efficient active vision system that can be used in numerous computer vision and control applications.

Part II describes CMOS image sensor design considerations for active vision applications. Specifically, these chapters discuss methods of determining regions of interest and achieving high dynamic range on the sensor.

Part I

System-Level Active Vision Design

Chapter 2

The Human Vision System

At the highest level of abstraction, the eye is a lens that focuses onto a retina. The retina is a mosaic of two types of photoreceptive cells: rods and cones. Rods are sensitive to most visible wavelengths and are used for low-light vision. Cones are sensitive to details and colour, and are responsible for high-acuity vision. The fovea, shown in Figure 2.1 is the centre of the retina; it contains the highest concentration of detail and colour-sensitive cones, and is used for fine vision and tasks such as reading [18].

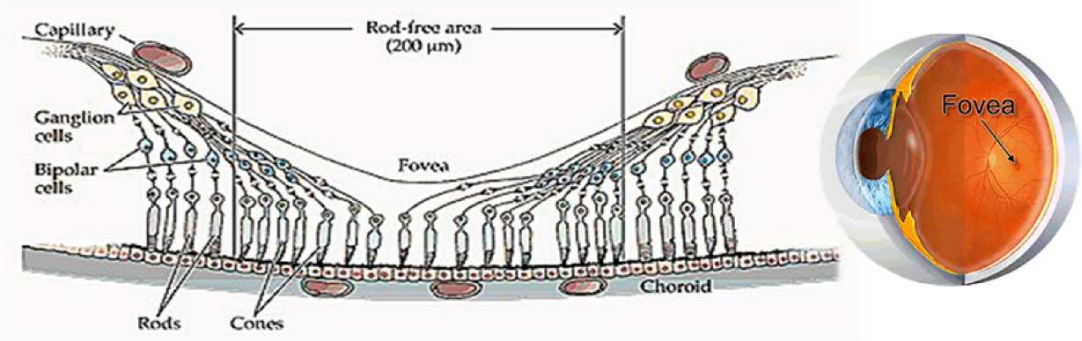


Figure 2.1: The fovea, located at the centre of the eye [19].

The fovea covers approximately 3% of the field of view, which is roughly equivalent to the area of a thumbnail held at arm's length [20]. Figure 2.2 shows that the concentration of cones peaks at the fovea, and drops drastically in the regions away from the centre. While the fovea is

rich in cones, the peripheral areas are filled with rods, which detect light under low illumination. The eye can only resolve spatial detail with high acuity at the fovea; thus the fovea must be redirected around the visual field in order to inspect all the pertinent points of interest.

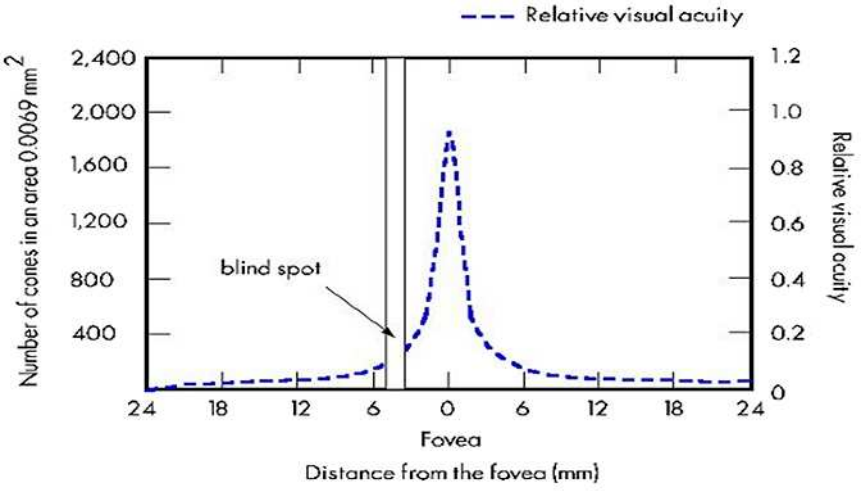


Figure 2.2: Visual acuity distribution in the eye [19].

2.1 Photoreceptor Concentration versus Acuity

In the early 1990's, Cha conducted a series of experiments to determine the relationship between number of pixels (in a visual prosthesis) and human visual acuity. He concluded that an array of 600 stimulation points implanted in a 1cm^2 area would suffice for 20/30 visual acuity[1], providing adequate acuity for reading [22] and visually guided walking [23]. He further found that a subject's reading rate increased with number of pixels in a contained viewing window, suggesting that the increased density of pixels improved acuity and allowed quicker recognition of characters [22]. At the centre, the fovea is estimated to contain approximately 180,000 cones per mm^2 [24]; this is the concentration of photoreceptors that provides our highest level of acuity. However, Cha's

[1]In 1862, Hermann Snellen designed the Snellen Chart to quantify visual acuity. By definition, a person with "normal vision" should be able to (barely) resolve the letters on the eighth line of the chart from a distance of 20 feet; this denotes 20/20 vision. The same person should be able to resolve the sixth line (the 20/30 line) from a distance of 30 feet. A person who can barely resolve the sixth line from a distance of 20 feet is said to have 20/30 vision.[21].

experiments show that we are not crippled by a lower concentration of photoreceptors; we simply are not able to resolve as ably. A vision system with reduced photoreceptors can therefore prove functional and provide adequate acuity for simple tasks, such as basic shape recognition and feature detection.

2.2 Dynamic Range

Human vision is able to adapt to a wide range of luminance levels. Although the eye will initially become saturated when introduced to strong light (and similarly have trouble seeing in the sudden absence of light), it can quickly adjust to the scene's light conditions. Once adjusted, the eye is capable of distinguishing over nine orders of magnitude of luminance [25].

The cones in the fovea are not only responsible for high-acuity vision, but are also used to view details in the presence of high illumination. Their counterparts, rods, dominate the peripheral regions of the eye and are effective at distinguishing information under low illumination [26]. The combination of cone and rod sensitivities gives the eye its ability to perceive details in both dark and light regions of the same scene.

2.3 Eye Movements

Eye movements serve many purposes, including redirection of attention, gaze stabilization, and refresh of visual signals on the photoreceptors (cones and rods saturate if exposed to the same signal for overly long [20]). There are two main types of eye movements that are associated with attention: smooth pursuit and saccade.

2.3.1 Smooth Pursuit

When an object moves within the visual field, the eye fixates onto the target and follows its path by smoothly matching the motion. When the object nears the boundaries of the field of view, the neck moves the head to maintain the object within the field. The eye continues to fixate on the object, stabilizing it on the fovea.

2.3.2 Saccade

From the French word saquier, "to twitch" [27], saccades are quick, jerky eye movements that direct the fovea's attention from one point of interest to another. The eye makes approximately 3 saccades per second, fixating on each point of interest for 300ms at a time [18]. The brain acquires an understanding of the scene through a series of saccade-fixation sequences. It then incorporates this information to construct a coherent representation of the scene [3]. This is all transparent to the consciousness, as the viewer is only aware of seeing the entire scene instantaneously. In order to prevent image blur and jerky transitions between fixations, the eye experiences saccadic suppression during the jumps: that is, the eye suppresses the optical signal and is essentially blind during the eye movement [26].

2.4 Mapping Eye Movements

When a new scene is introduced to the eye, certain stimuli will 'catch the eye's attention', marking some regions in the scene as more conspicuous than others. The measure of a stimulus' conspicuity depends on many factors, including the context of the scene, the object's proximity to other stimulating sources, the objective task, and the personal preferences of the subject. There are also some aesthetic features that might be considered intrinsically stimulating, such as bright objects, high contrast, etc. The orientation of an edge might also affect an object's conspicuity.

The brain's cortex is responsible for the evaluation of sensory input. Visual sensations trigger signal responses from the rods and cones. These signals are fed to the feature extraction engines in the cortex, which map regions of salience in the scene. In neural psychology, the composite of the various feature maps is termed the 'saliency map.' The brain uses the saliency map to determine the regions in the scene that require foveal attention. Figure 2.3 shows Fulton's block diagram, in which he outlines the relationship between the various sensory centres and the processing of their inputs. The brain retains a database in its short term memory to preserve some basic knowledge of the scene [24].

Studies of human visual attention suggest two visual search patterns: bottom-up ("preattentive") and top-down ("attentive") analysis [28]. In a bottom-up search strategy, foveal attention

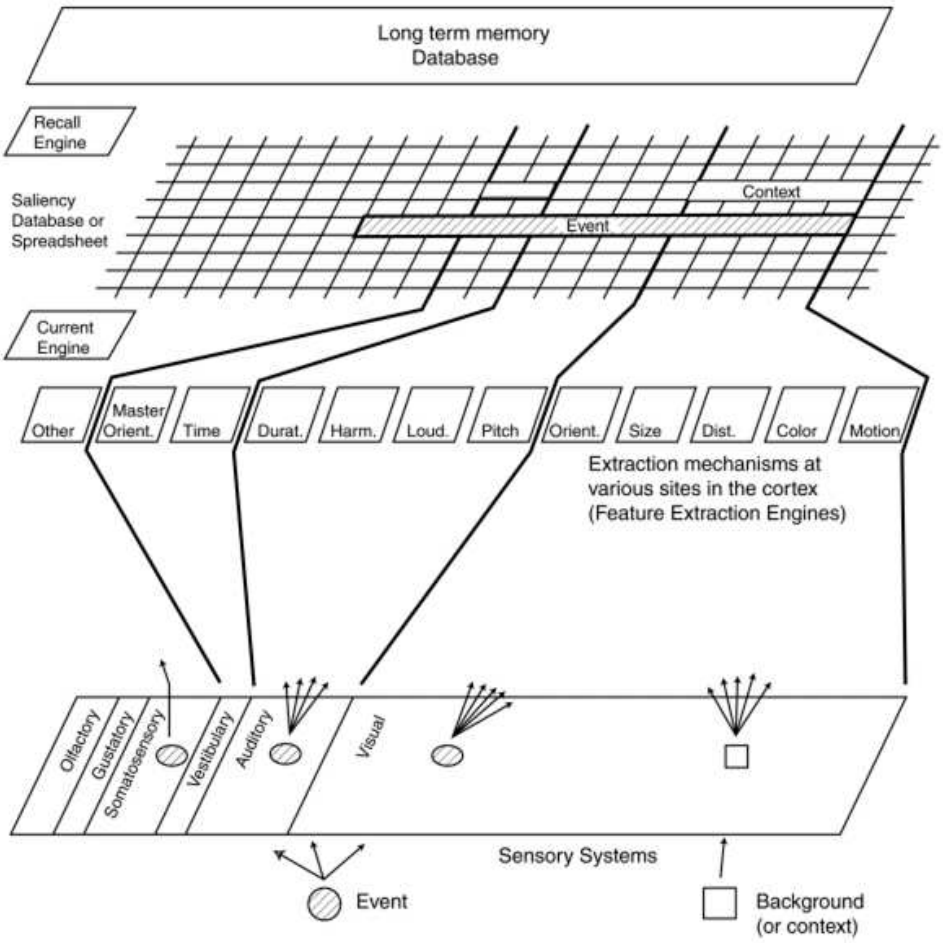


Figure 2.3: Mapping saliency in the brain, with emphasis on vision [24].

is directed at locations containing generic stimulating features, such as color, orientation, and intensity [29]. The scanpath results from the salient conspicuity of each location in the scene.

A top-down search strategy is task-driven, where the vision system hunts for objects that are relevant to the specific scenario. Consider the illustration depicting the foveal attention of a subject who intends to boil water, Figure 2.4. Each dot indicates a unit of time that the fovea spends on a location. It makes sense that the fovea searches out only the details relevant to the task, and leaves the "less important" information to low-acuity detection regions of the eye. This is a top-down, task-specific search [18].

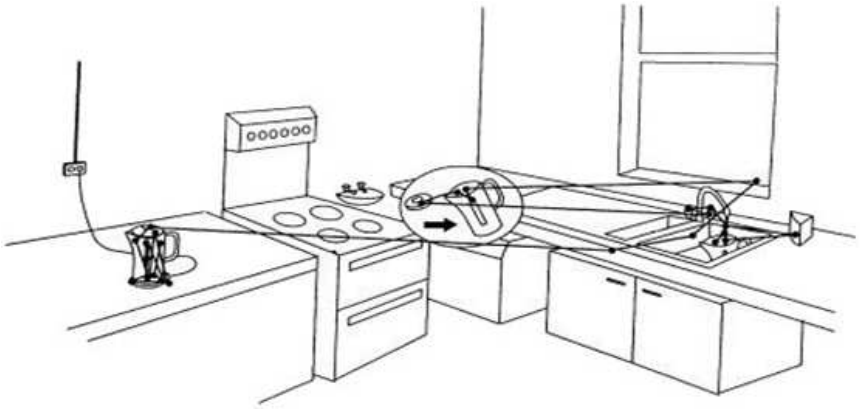


Figure 2.4: A top-down survey of a scene, made by a person who has the intention of boiling water [18].

Now consider Figure 2.5, which shows eye movements recorded while studying a photograph of an Egyptian bust. Again, each dot represents a unit of time. Note that the subject's analysis of the statue does not require foveation to every point of the face; the subject is satisfied by only attending to the eye, nose, mouth, ear and outline areas. This is a bottom-up search, where the subject only foveates to those areas that are conspicuous, or those features which the subject is conditioned to consider a priority. From only a small percentage of the visual field captured in high-acuity, the subject is able to reconstruct a useful understanding of the object. This image is part of a collection of recorded eye movements measured during an experiment conducted by Yarbus. He noted that the attention of the fovea is reserved for those elements that contain "essential information" necessary for the perception of the scene. The "less essential information"

is ignored by the fovea and obtained by the lower resolution periphery [3].

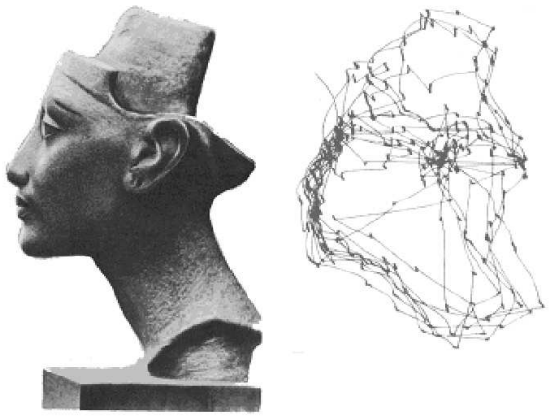


Figure 2.5: A bottom-up survey of the photograph of a statue [3].

Chapter 3

Mike: Prototype Design

3.1 Project Goals and System Requirements

The Mike project's objective is to realize a closed-loop, selectively attentive, visual data acquisition system that can respond to the image data in real time. Consider again the example of the Mars Rover [7]. Although Mike is not intended specifically for the Rover, the application of a visual navigation system for an autonomous vehicle provides a useful application for which to define the system requirements:

1. low processing time and power;
2. minimal data transmission;
3. real-time access to data presented in a useful form; and
4. fault tolerance against minor damage to potentially vulnerable system components.

3.2 The Need for Reduced Resolution

Consider a computer vision system with a SXGA monochrome sensor (1.3 MPixels). For real-time use, assume that the system operates at a frame rate of 30fps, using 3x3 pixel kernels (structuring elements) for the image analysis. The image analysis would therefore require image processing on

350 Mpixels per second. The transmission of 30 uncompressed frames containing 8-bits per pixel would require a communications rate of 315Mbit/s. Although USB 2.0 can transmit at 480 Mbit/s [30] and IEEE 1394a (FireWire) can transmit at 400Mbit/s [30], the Mars Microrover's radio has an effective maximum data transmission rate of 2400bit/s using its 100mW RF transmitter [31]. Employing Jet Propulsion Laboratory's incremental cost-effectiveness ratio (ICER) compression scheme [32] to compress the image, the system would still be required to transmit 26.24Mbit/s; this is 4 orders of magnitude beyond the capabilities of the Microrover's radio. It is therefore imperative to limit the amount of information placed on the transmission stream. This can be achieved by either reducing the effective field of view to a smaller window, or reducing the resolution of the image. The former approach constrains the system's vision to a small physical area. While this approach will allow the system to respond to its visual field in real-time, the response is only to a very small portion of the physical space. The latter approach maintains a wide-angle view of the scene, but reduces the density of pixels that encodes each physical region. This will diminish the level of detail describing physical objects, but will maintain the general shapes of the objects. It is comparable to taking a low-level detailed description to a higher level of abstraction. This work proposes a system that analyzes a reduced resolution image to determine the salient regions that warrant full resolution image capture; thereby conserving both processing time and power, and efficiently utilizing transmission resources.

3.3 Top-Level Architecture

Mike comprises four main subsystems: the camera (the 'eye'), the servomotor (the 'neck'), the control block (the 'brain'), and the saliency map engine (which entails a combination of eye and brain functions). These four subsystems are shown in Figure 3.1.

3.4 The Eye

The camera is responsible for image acquisition. It contains an image sensor (an array of photo-sensitive pixels), a lens (that focuses light onto the sensor array), and an interface board (that sends commands to the sensor chip and reads the raw image data). The imaging chip converts

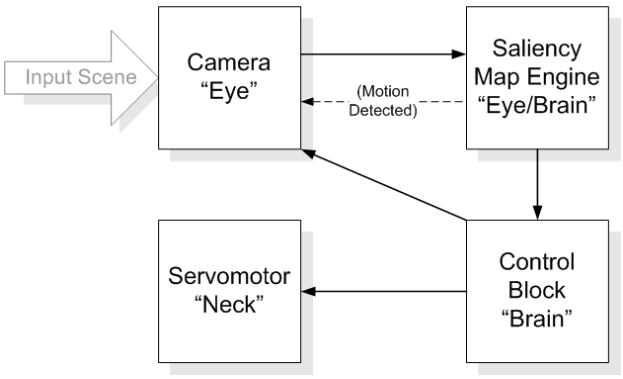


Figure 3.1: Block diagram of the system architecture.

analog signals (proportional to the incident light intensity) into digital representations. Therefore, an 'image' is a numerical matrix whose elements contain digital values proportional to the incident light.

There are two major types of solid state image sensors: charge coupled device (CCD) and complementary metal oxide semiconductor (CMOS). The two technologies differ in the method by which charge is transferred off of the sensor array. CCD image sensors boast higher fill factor and noise immunity, but CMOS sensors can be integrated with in-pixel processing circuitry [33]. This means that specific pixels, or groups of pixels, can be selectively addressed, thereby allowing for efficient subwindowing, fast frame readout, and programmability. The addressability of pixels is particularly useful for the Mike design because it provides subwindowing capability: the ability to read out a specific set of pixels, demonstrated in Figure 3.2. (A CCD camera may also claim to provide subwindowing readout, but in reality, specific columns cannot be isolated and thus entire rows must be read out; the subwindow would then be assembled external to the readout circuit.) Furthermore, the flexible readout of CMOS imagers can be exploited to generate low resolution images through decimation, where only pixels from a select pattern of rows and columns are read (such as every 2nd or every 4th pixel along each row and column).

The Mike prototype uses a commercial, programmable monochrome 1.3MPixel (SXGA) CMOS camera [34]. The camera connects to a 2.4GHz computer via an IEEE 1394a (FireWire) interface that transfers camera instructions and data at up to 400Mbps [30].

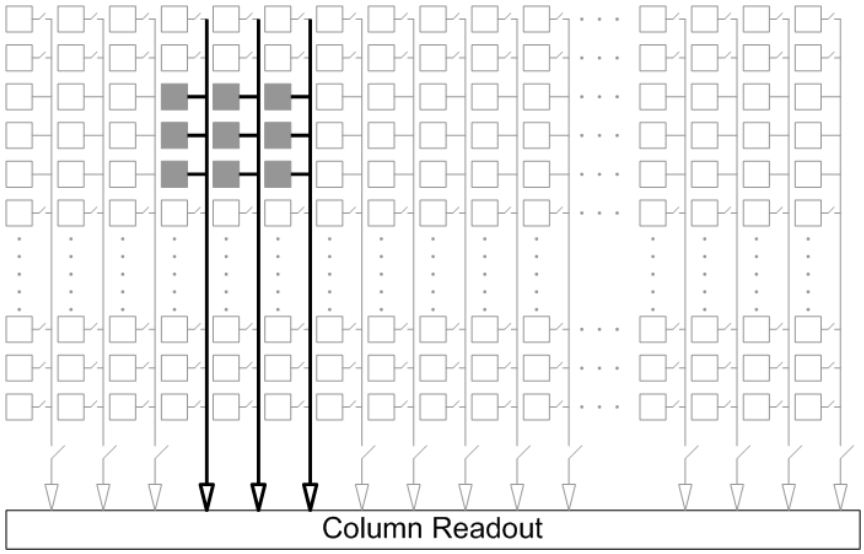


Figure 3.2: Subwindowing on a CMOS sensor array.

3.5 The Neck

The camera is mounted on a programmable pan/tilt servomotor, see Figure 3.3. Once the system has completed inspection of a static visual field, coordinates are sent to the servo to modify the camera's perspective of the scene. The servomotor can also be used for a wide-angle pursuit of a moving target or for stabilization of a moving/vibrating object in the camera's view.

3.6 The Saliency Map Engine

The term 'preprocessing' refers to the first-pass image processing that is used during early stage vision; its purpose is not to extract exact information from the scene, but rather to generate a raw primal sketch that represents the major objects and structures in the visual field [26]. In a bottom-up visual search, the vision system conducts a topographic survey of the scene, generating maps of conspicuous locations, (where conspicuity is a measure of the presence of a specific stimulus in that location). Several maps are generated; each map contains weighted salience information about a different feature or stimulus, such as corner and edge information. The maps are then

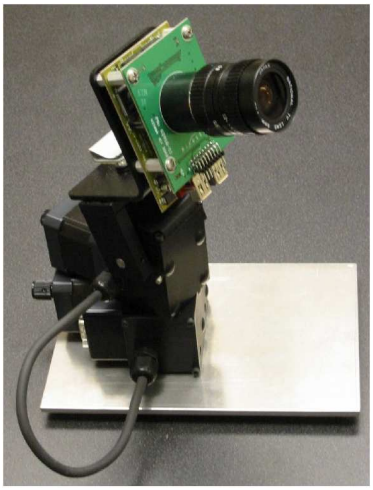


Figure 3.3: The Mike prototype consists of a commercial CMOS camera mounted on a programmable servomotor.

combined and tallied for overall salience in the space. A more detailed discussion on the saliency map is presented in Chapter 4.

3.7 The Control Block

The control block sends and receives commands and messages between the camera and the CPU, and between the servomotor and the CPU. Mike is a closed-loop active vision system that operates in two modes: scan mode and saccade mode, shown in Figure 3.4. (There is also a third mode, an adaptation mode, used to recalibrate the system in a situation where distortion is introduced and the lens cannot be easily replaced.) In scan mode, the camera is set to 1/4 SXGA resolution for a coarse sampling of the visual field. The low-resolution image acquired in this mode is 'preprocessed' to quickly identify regions of interest that require more detailed inspection at high-resolution. Specifics on the image analysis are described in Chapter 4. In saccade mode, the imager serially revisits the regions of interest with a small subwindow set at full SXGA resolution. This subwindow serves as the system's 'fovea.' Since the dynamic fovea is electronically guided, eye movements are not hindered by issues typically faced by mechanical systems, such as slow response, non-precise repeatability, and backlash. The control block also determines the pan

and tilt coordinates for the servomotor. After the camera has completed a round of scan and saccade, the servomotor is redirected to another angle to study a different field of view. If there are any foreground pixels that lie along a boundary of the frame (i.e. if an object straddles the frame border), then the servo is programmed to redirect the camera perspective such that more of that object is contained in the field of view. Since there is no information on how far the camera needs to move in order to view the entire object, the degree of motion is a random value. Foreground objects are determined through binarization of the image. Therefore, if any foreground pixels lie along the frame boundary, the servo is given the command to continue 'looking' in that direction. If there are foreground pixels along more than one boundary, then the final pan and tilt coordinates of the servo take into account both boundaries. The boundary with the largest number of foreground pixel dominates the final choice of coordinates. If there are no foreground pixels along any of the boundaries, then the control block randomly selects a new set of pan and tilt coordinates.

3.7.1 Scan Mode

Mike employs a bottom-up search strategy to determine potential regions of interest during scan mode. The camera captures a 1/4 SXGA survey image of the scene and extracts the locations of salient areas based on this image. The output of this mode is the saliency map: a list of the most conspicuous points in the scene (a detailed discussion on this topic is provided in Chapter 4).

3.7.2 Saccade Mode

The saccade mode accepts as input the ordered saliency list generated by the scan mode. The camera serially traverses the list, capturing a fully resolved 32x32 subwindow image (3% of full view) at each of the salient coordinates. The final output of the saccade mode is a reconstruction of the scene, where only the salient areas are detailed in high-resolution. After the saliency list is exhausted for the current scene, the motorized servo shifts the camera's position for a new view.

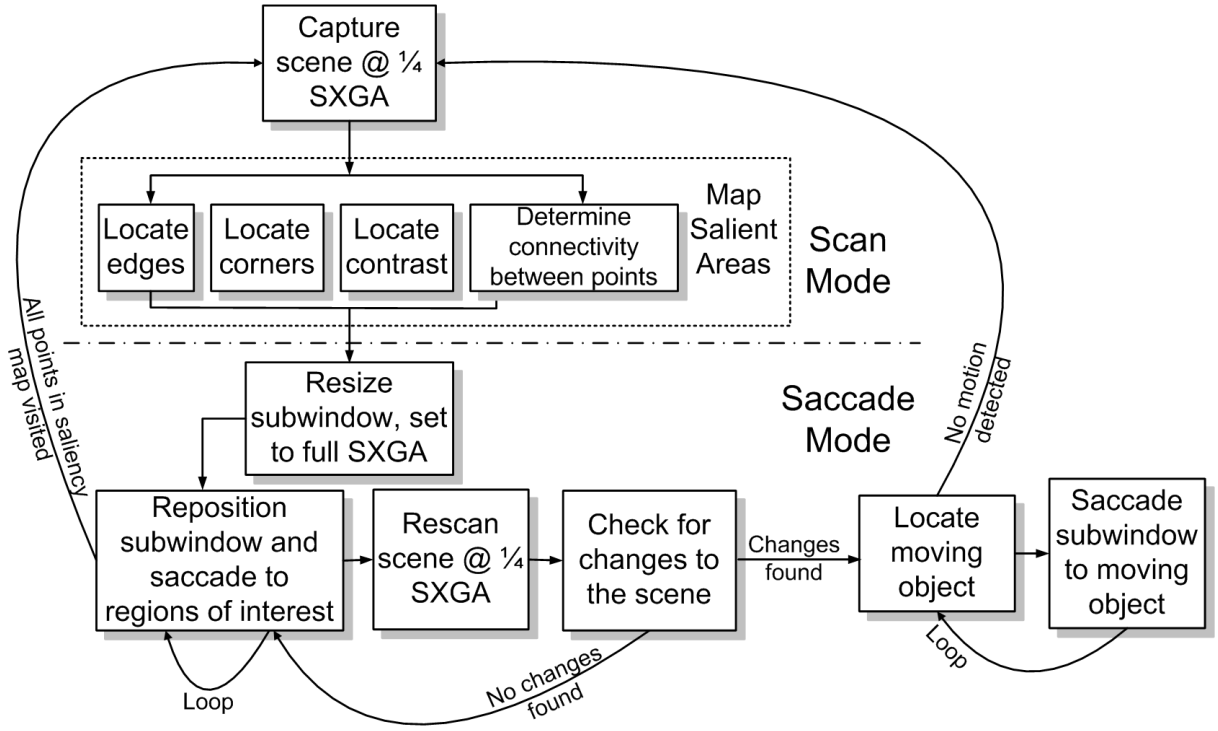


Figure 3.4: Flow diagram of the scan and saccade operation modes.

Motion Detection and Tracking

Since the fovea and periphery occupy separate parts of the biological eye, the eye can quickly detect motion concurrently to foveation. However, on the camera, both low-resolution and high-resolution image captures share the same sensor space, and the camera's firmware does not allow for multiple resolutions to be applied simultaneously. Thus, low-resolution scan and high-resolution fixation must be performed separately. During high-resolution fixation, Mike is essentially 'blind' to all areas of the visual field other than the region of interest attended by the foveal subwindow. Thus motion detection must be performed in saccade, rather than scan, mode. Every five saccades (an optimal value obtained through experimentation), the system recaptures a quick low-resolution image of the full view and compares it to the original survey image acquired during scan mode. Motion is defined here simply as a detectable difference between the two low-resolution images. If a change has occurred, the system suspends attention to the locations marked by the saliency map and attends to the changed areas. Otherwise, the subwindow continues through the list of saliency coordinates. Multithreading the software would reduce delays in saccade execution during motion detection.

Motion pre-empts all other salient stimuli. When motion is detected, saccades are reserved for "vital" regions of interest that provide cues useful for tracking and identifying the object. This necessarily reduces the number of saccades and allows the system to keep pace with the moving object, as each foveation constitutes one frame readout. Using data obtained from foveating the vital regions, the system computes an estimate of the moving object's position relative to the camera. In order to improve the speed and accuracy of tracking, the system employs a simple predictive algorithm based on a history of the object's previous locations (more details on motion detection are provided in Chapter 6).

Locally Adaptive Exposure Times to Salient Regions

Most photographs of scenes with a range of light levels suffer from either over-exposure (of bright regions), or underexposure (of dark regions). A camera's exposure time can be optimized for either the bright or the dark regions, but not both. This results in images with poor dynamic range. The PixeLink camera contains a CMOS image sensor with an average of 50dB dynamic

range for a contant exposure setting [34]. This is 6.5 orders of magnitude less than the dynamic range of a human eye [25].

A simple, but extremely useful application of the Mike camera is to use it as a high dynamic range system. As the subwindow fixates on each salient region, the camera's exposure time is adjusted for the light conditions local to that region. Therefore, the salient points in the scene are not only presented in high resolution, but are properly exposed. Wilburn uses a similar technique to adjust the dynamic range of a wide-angle scene by capturing the scene using an array of cameras; each camera's exposure time is specifically adjusted for its narrow field of view [35]. Larson also constructs a high dynamic range photograph using a composite of 16 photographs taken at different exposures [25]. In Mike's implementation, the local exposure time adjustment is reserved only for the salient regions in the scene, shown in Figure 3.5.

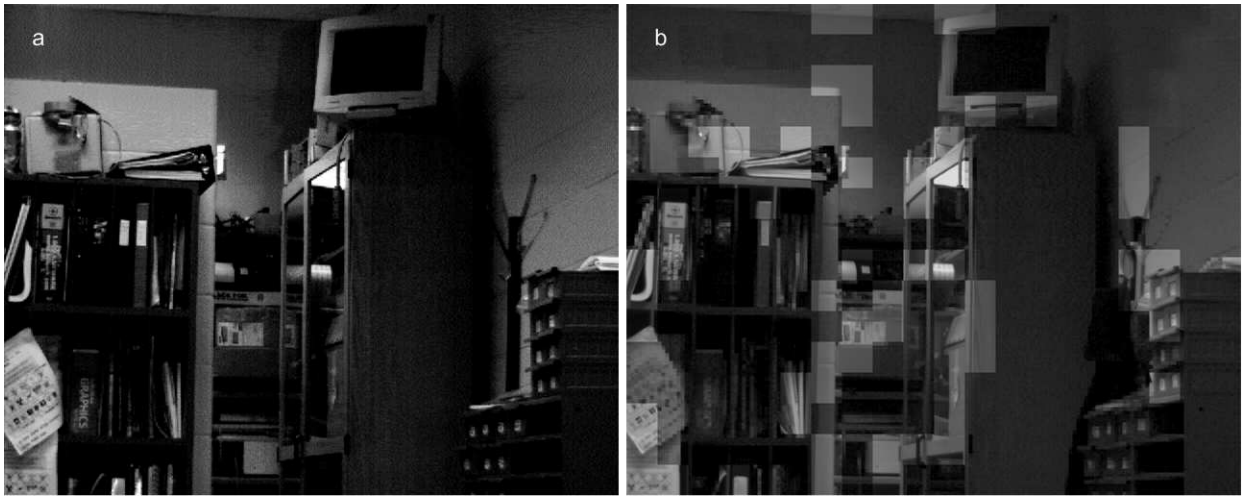


Figure 3.5: a) Image taken with one global exposure time and one resolution. b) Image taken with two resolutions and locally optimized exposure times in the salient regions.

3.7.3 Adaptation Mode

In addition to the two operational modes, (scan and saccade), the design includes a third mode for fault tolerance: the adaptation mode. This mode functions outside of regular operation. It is intended for applications that require a knowledge of the relationship between the image

information and real-world locations, such as a navigation guidance system. Mike enters this mode in a situation where the camera's optics are exposed to harsh conditions and cannot be easily repaired or replaced. The adaptation scheme exploits the knowledge of the image sensor's electronic coordinates relative to the camera's mechanical movement, to develop an empirical distortion model of the image formation process. This allows Mike to dynamically adapt to changes in its image quality. Details on the determination of the lens distortion model are presented in Chapter 5.

3.8 System Parameter: Resolution for Survey Image

The Pixelink camera is able to decimate an image by 2 or by 4 [34]. Decimation is the process whereby only every other pixel is read (in the case of decimation by 2) or every fourth pixel is read (in the case of decimation by 4). Figure 3.6 shows a decimate by 4 readout, which effectively produces a 1/4 SXGA image. This process retains the general information contained in most images, while decreasing the density of pixels that record the physical details of the scene. Although this particular camera unit only allows two decimation settings, the image can be further decimated in software prior to processing. Therefore, this section will investigate the optimal resolution setting for the survey image used in Mike's scan mode.

Figure 3.7 shows the effects of decimation. Decimation introduces aliasing and false corners. The aliasing effect is apparent in the rooftop surface of Figure 3.7 b, c and d, and in the parallel logs of the cottage facade in Figure 3.7 c and d; the steeper the slopes, the stronger the aliasing effect. The false corners are also most pronounced in the 1/16 resolution image, but can also be noted along the right edge of the cottage roof in the 1/4 resolution image.

Figure 3.8 plots the top 100 salient points that result from analysis of the four differently resolved images. Four features are considered in the determination of saliency: corners, edges, intensity contrast, and connectivity (the connection of salient points belonging to the same object). These features will be discussed in further detail in Chapter 4. Of the four features, the corner detector is most affected by the loss of resolution. In general, the outlines of large objects are maintained even when the image is decimated, and the general grey level of object regions are also preserved, thereby providing similar results in the intensity contrast and connectivity

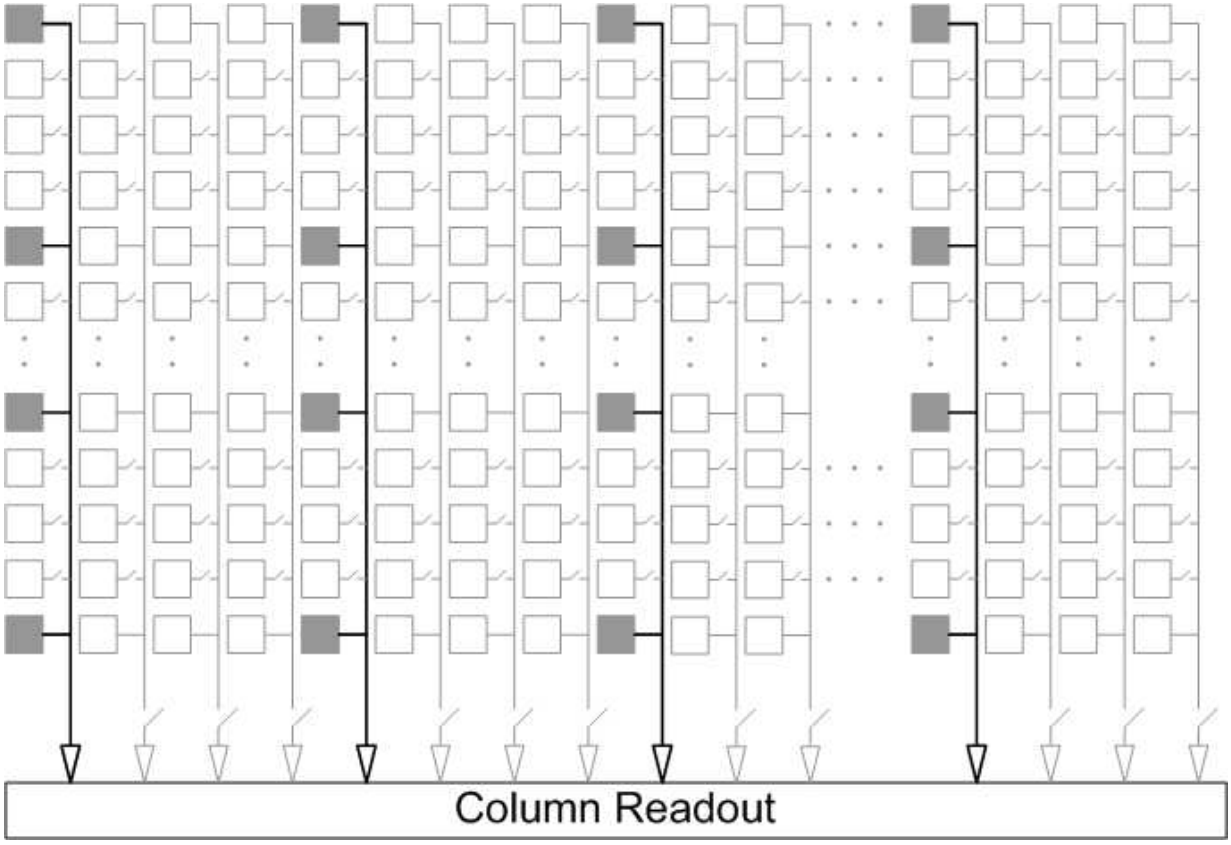


Figure 3.6: Decimation in a CMOS image sensor array.

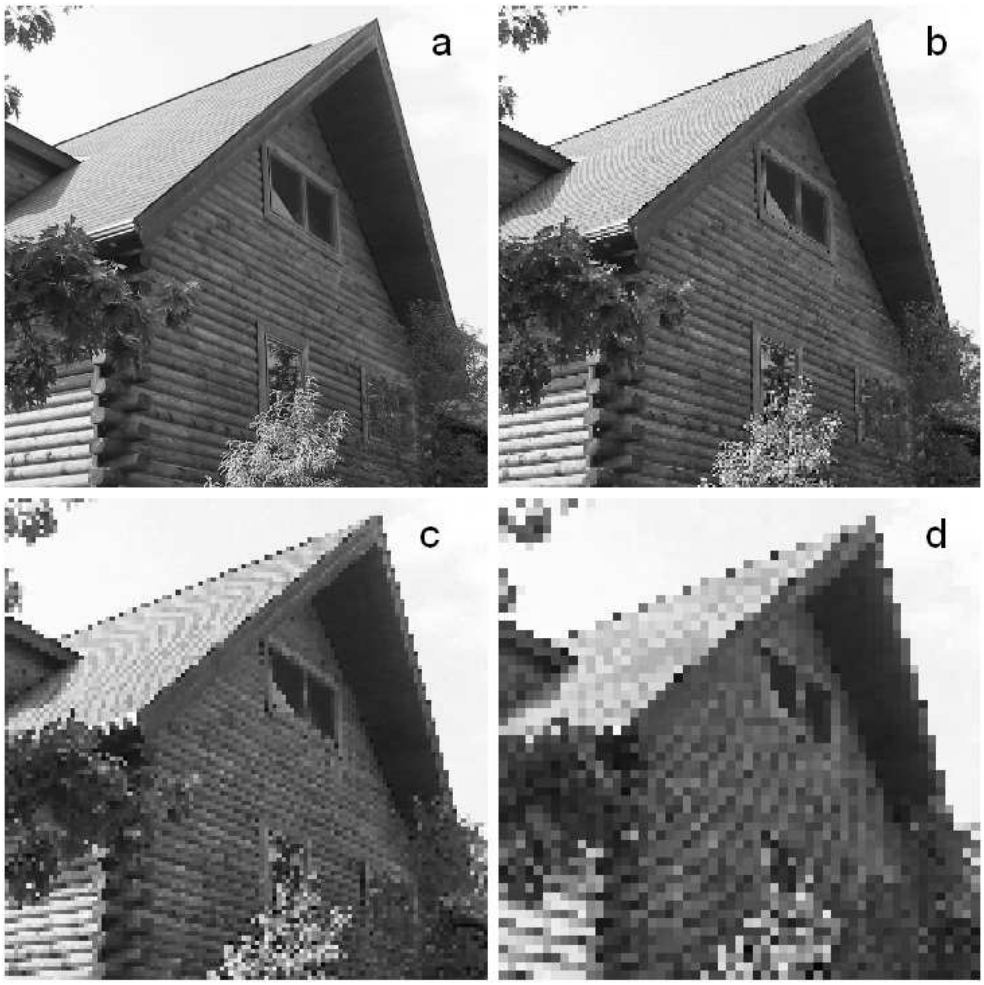


Figure 3.7: a) SXGA resolution (original image). b) 1/4 SXGA resolution (obtained through decimation in software). c) 1/8 SXGA resolution (obtained through decimation in software). d) 1/16 SXGA resolution (obtained through decimation in software).

routines. An excessive reduction of the image resolution can have one of two consequences: i) the introduction of numerous false corners, or ii) the removal of corner detail, such that the corner detector is unable to function properly. This is the case in Figure 3.8, where the corner detector is unable to return many corners from analysis of the 1/16 SXGA image. Therefore, the information from the contrast intensity map determines saliency in the bright sky, rather than the cottage rooftop. It can also be seen, however, that the full resolution image focused almost entirely on the trees, and almost completely ignored the house.

An unexpected but interesting consequence of false corners (from the pixelation in low resolution images), is that the false corners allow the corner detector to track curved surfaces. Figure 3.9a-d show the effects of decimation on a curved object. Normally a curve detection algorithm would require fitting object shapes to pre-defined curves, using a computation intensive routine such as the Hough algorithm [36]. However, since the goal is to draw attention to the curve, and not characterize its properties, it is enough to simply classify the curved surface as 'salient'. This is certainly achieved if the curved surface exhibits false corners. Therefore, there is no need to implement a special curve detector when the curves are represented as corners. For comparison, Figure 3.9e-h show the effects of decimation on straight edges at different orientations. Edges that are parallel (or within a few degrees of) the image axes do not exhibit false corners under decimation; edges that are at an angle with respect to the axes do suffer from the appearance of false corners, but not as severely as curved edges.

The choice of resolution is an important design parameter, as the overall number of pixels directly affects the time required for processing the image matrix. Figure 3.10 shows the relationship between image resolution and processing time of the corner detection algorithm run in MatLab (see Appendix B for the implementation). The processing time decreases exponentially with number of pixels in the image: the 1/4 resolution contains 1/16 the number of pixels (reduction by 4 along the rows and reduction by 4 along the columns), however analysis of the corners in the 1/4 resolution image takes a factor of 20 less time than processing of the full resolution image.

In Figure 3.8, analyses of the 1/4 and 1/8 resolution images provide similar results. This however, may not be the case for all types of scenes. To balance the tradeoff between speed and

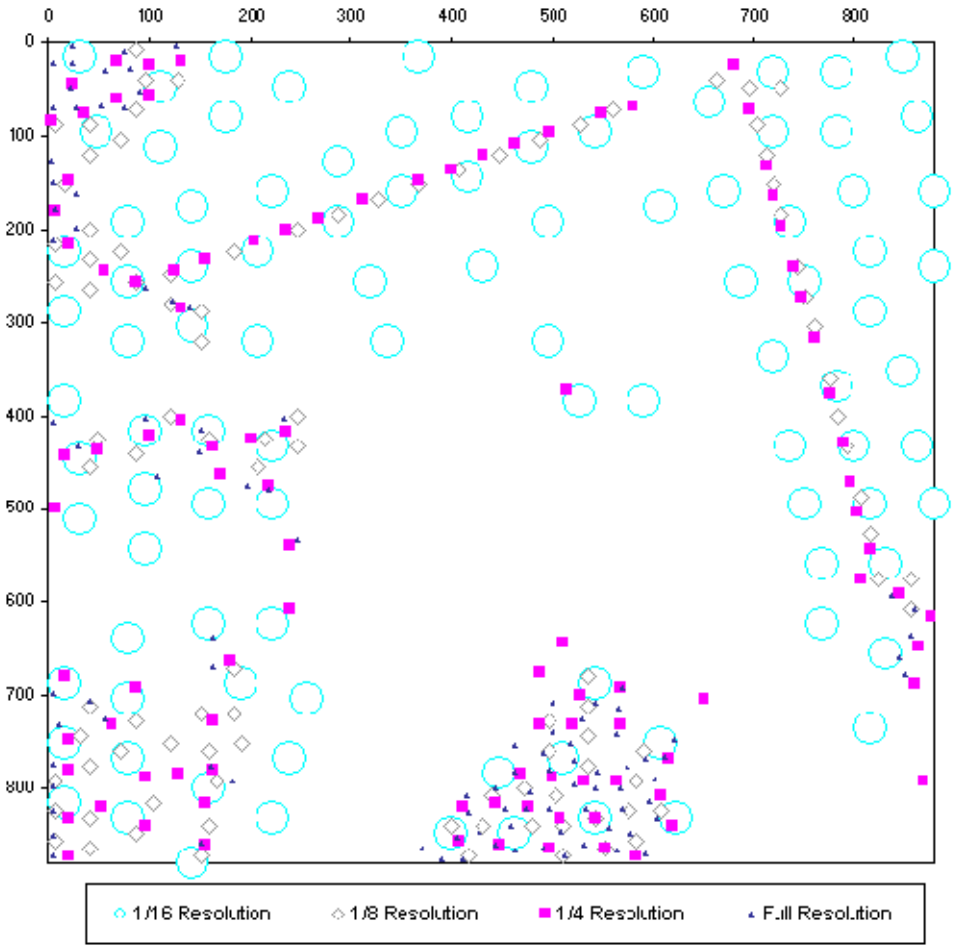


Figure 3.8: Spatial location of the top 100 salient points determined through analysis of the input image at: full resolution, 1/4 resolution, 1/8 resolution, and 1/16 resolution. The coordinates of the salient points from the 1/4, 1/8 and 1/16 resolution images were multiplied by 4, 8 and 16 respectively in order to map the 'equivalent' coordinate in the full-resolution space. The size of the markers represent the area of uncertainty due to this mapping process. For example, the markers for the 1/16 resolution salient points occupy a 16x16 pixel space.

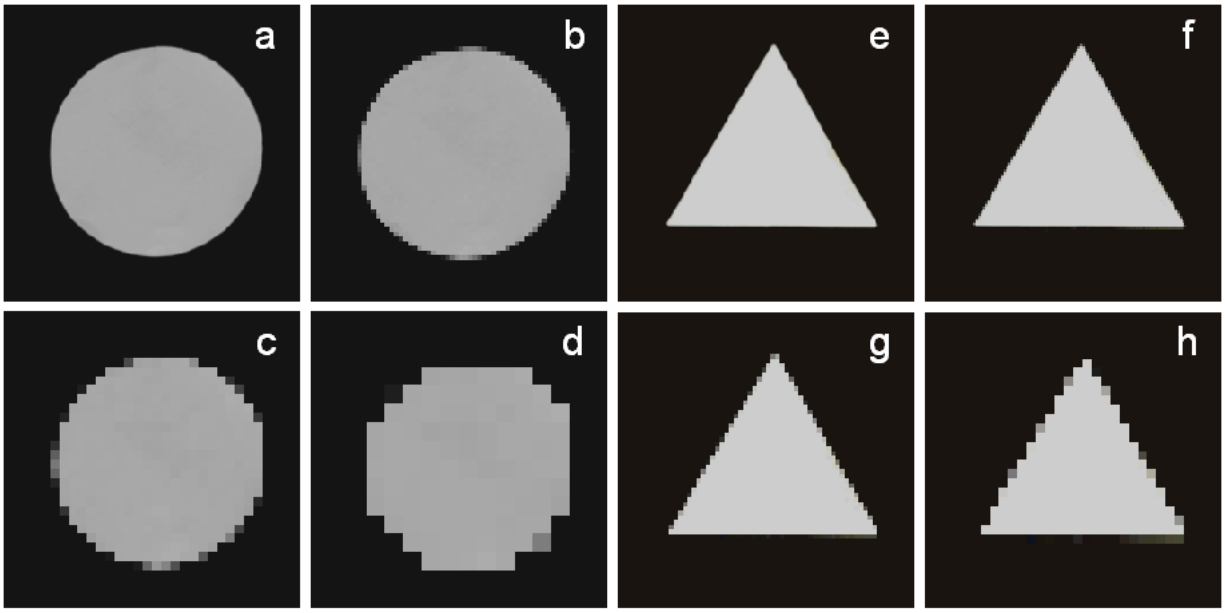


Figure 3.9: Pixelation effects on specific geometries. a-d are the effects of pixelation on a curved shape in SXGA, 1/4 SXGA, 1/8 SXGA, and 1/16 SXGA resolution, respectively. e-f are the effects of pixelation on a triangular shape in SXGA, 1/4 SXGA, 1/8 SXGA, and 1/16 SXGA resolution, respectively.

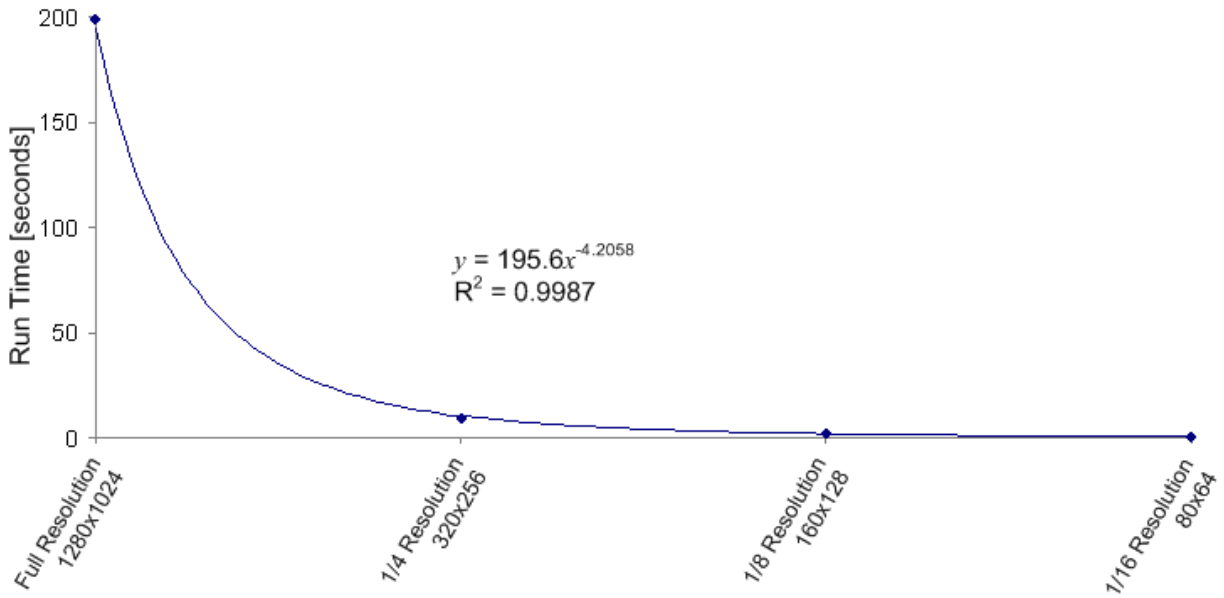


Figure 3.10: Feature extraction execution times vs image resolution

quality of results from the feature detectors, 1/4 SXGA is selected as the resolution used in the scan image. However, for applications that require very quick and less accurate analysis of survey scenes, 1/8 SXGA would also suffice for the low-resolution feature detection. Too much detail is lost in 1/16 SXGA images, therefore that is not a recommended alternative for the scan mode resolution.

3.9 Foveal Grid

Although the PixeLink camera allows the subwindow to be programmed anywhere in the sensor array, it would be inefficient to saccade to locations that overlap regions that have already been visited by previous saccades. Therefore, to prevent the system from repetitively capturing overlapping subwindows, Mike divides the sensor into a pre-defined grid of 32x32 pixel sectors. The fovea can therefore be any integer multiple of a 32x32 pixel sector. Sectors containing the coordinates of salient points listed in the saliency map are visited during saccade mode. Once a sector has been visited, it is inhibited from further visitation until the next round of saccades.

Chapter 4

The Saliency Map

This chapter discusses the saliency map engine in the Mike system. The saliency map charts the topology of the salience of a scene. It is a weighted tally of the conspicuity of the scene based on four criteria: corners, edges, relative brightness/darkness, and connectivity (to other salient regions).

4.1 A Measure of Conspicuity

The conspicuity of an object and its features is highly subjective to its surroundings. A blue Smartie in a bowl full of multi-coloured Smarties is nondescript, whereas a blue Smartie in a bowl full of red cinnamon hearts is conspicuous. There are several visual features that might be considered conspicuous, and these are often cues that contribute to the cognition of objects and scenes. Such features include shape, texture, contrast (to surrounding area), intersections of edges (corners), repetitive patterns, edge orientations, colour, and general relationship to the surrounding area (the object's similarity to its neighbours) [26]. The visual stimulus that provides the most relevant information depends on the scene. Considering again the example of the blue Smartie in the bowl of cinnamon hearts, the most conspicuous feature would likely be colour, followed by shape. However, in a scene full of various wooden building blocks, corner detection would probably be the most useful, and the colour feature would provide very little distinguishing information between different blocks. Without an a priori knowledge of the contents of a scene,

it is difficult to know which type of feature would provide the most relevant information.

4.2 Implementation of the Saliency Map

Mike's saliency map has two implementations: in MabLab (for experimental simulations), and in C++ (to interface with the camera). Both sets of code are provided in the Appendix. The saliency map is a weighted tally of a combination of various features: corners, edges, intensity contrast, and connectivity with other salient points in the local neighbourhood. Corners are not strictly the intersection of two lines; rather, 'corners' refer to a sharp pattern in intensity change in the image. Contrast intensity is the deviation of a pixel's intensity value from the mean grey level of the scene. 'Connectivity' categorizes pixels with other pixels that potentially belong to the same shape or object. Edges are the outlines of shapes. The detection of these features is discussed in more detail in subsequent sections.

Consider the six figures presented on the following pages (Figures 4.1-4.6). They show a variety of greyscale scenes and their detected features. These figures are generated by the MatLab code presented in Appendix B. Their input images are encoded in 256 (8-bit) greyscale format with 1/4 SXGA (320x256) resolution, which is the resolution used in Mike's scan mode. Part a shows the most prominent corners detected in the scene, part b shows object edges, part c shows the intensity contrast, and part d shows the segmented shapes. Part e shows the three-dimensional plot of the combined feature maps, depicting the salience-magnitude. For example, in Figure 4.3, Dr. Hornsey's right hand is the most salient point in the scene. These many examples cover a variety of scenes to demonstrate the determination of salience in different types of scenes that contain different types of objects and features.

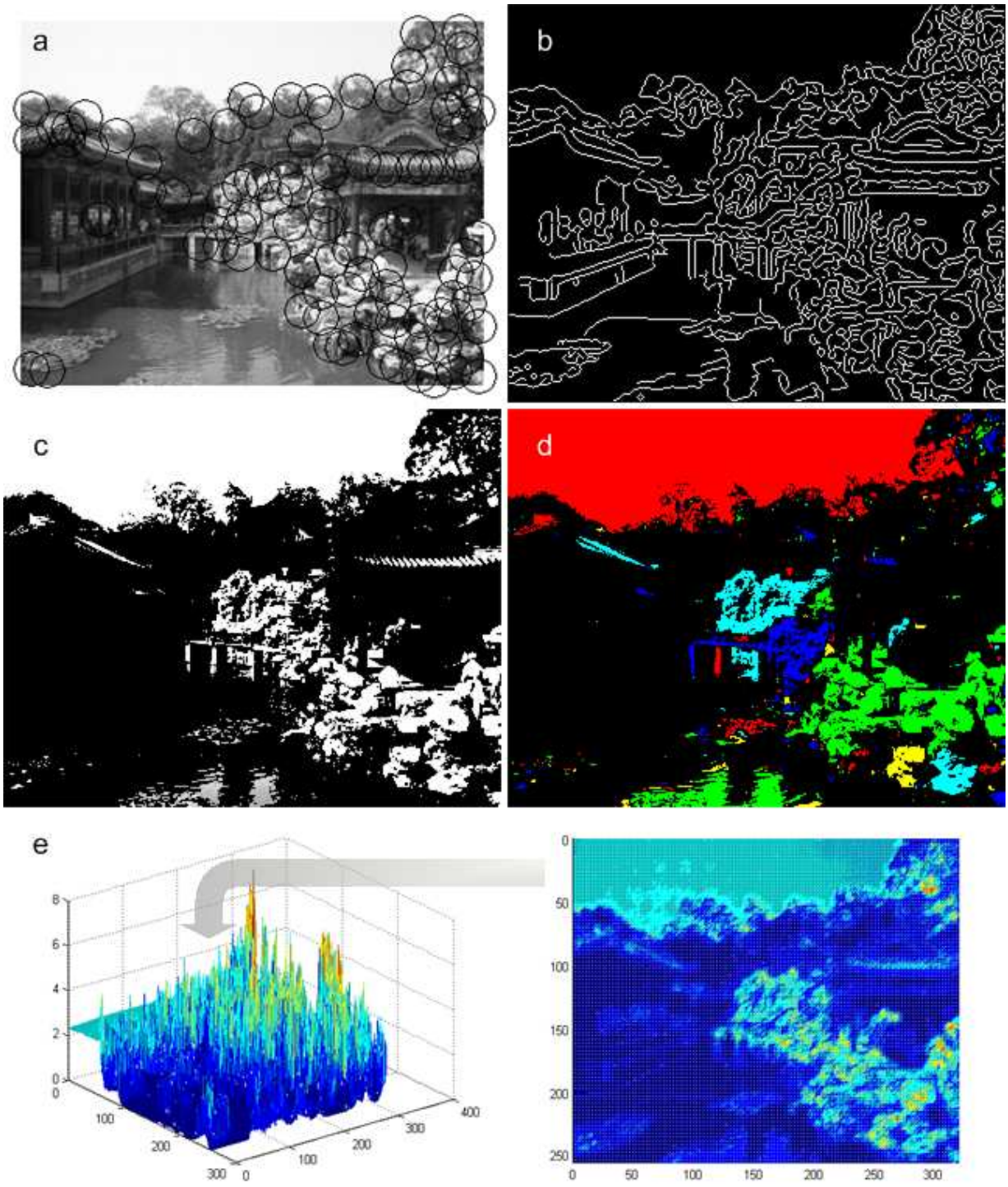


Figure 4.1: Bei Hai Garden in Beijing, China. Parts a-d display the output of various feature detectors: a) corner, b) edge, c) intensity contrast, d) connectivity (the different segments on the image show the different detected objects). Part e) shows the 3D mesh plot of the combined saliency map. The most salient point in this scene is located at pixel (296,41), which is near the top right.

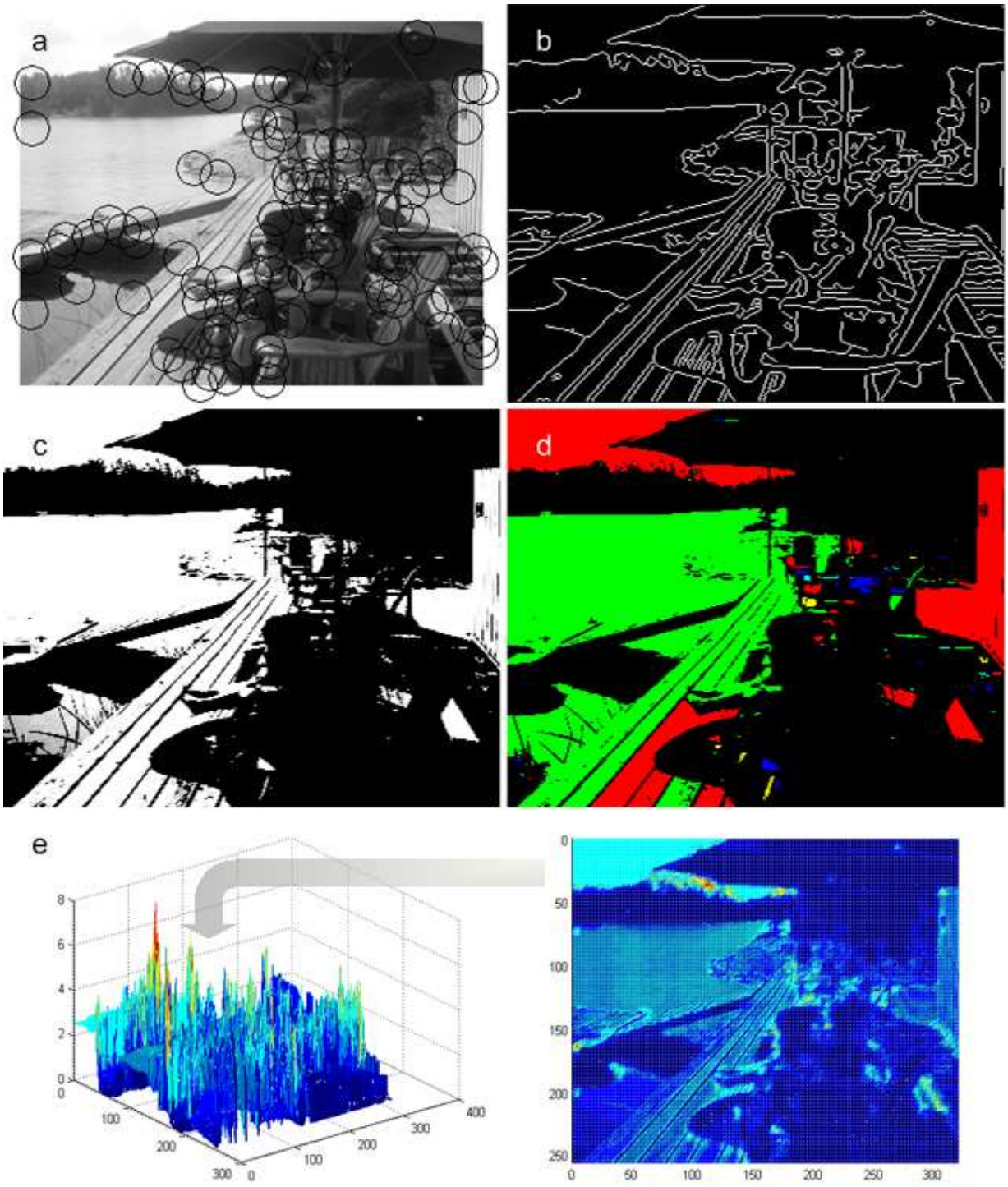


Figure 4.2: Docks at the Carr Cottage in Georgian Bay, Ontario. Parts a-d display the output of various feature detectors: a) corner, b) edge, c) intensity contrast, d) connectivity (the different segments on the image show the different detected objects). Part e) shows the 3D mesh plot of the combined saliency map. The most salient point in this scene is located at pixel (112, 37), which is near the top of the image.

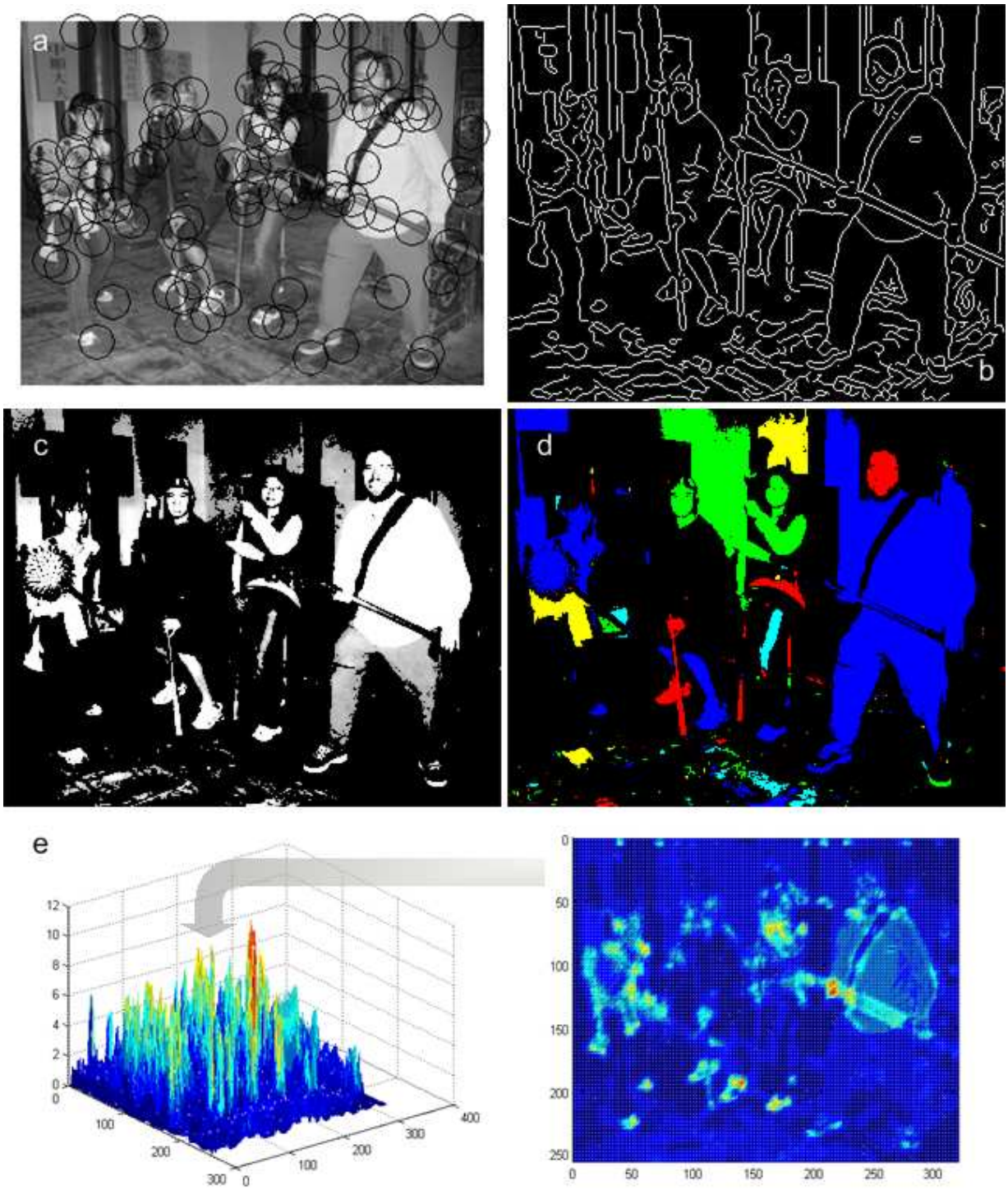


Figure 4.3: Armoury at Mu Mansion in Lijiang, China. Parts a-d display the output of various feature detectors: a) corner, b) edge, c) intensity contrast, d) connectivity (the different segments on the image show the different detected objects). Part e) shows the 3D mesh plot of the combined saliency map. The most salient point in this scene is located at pixel (138,194), Dr. Hornsey's hand.

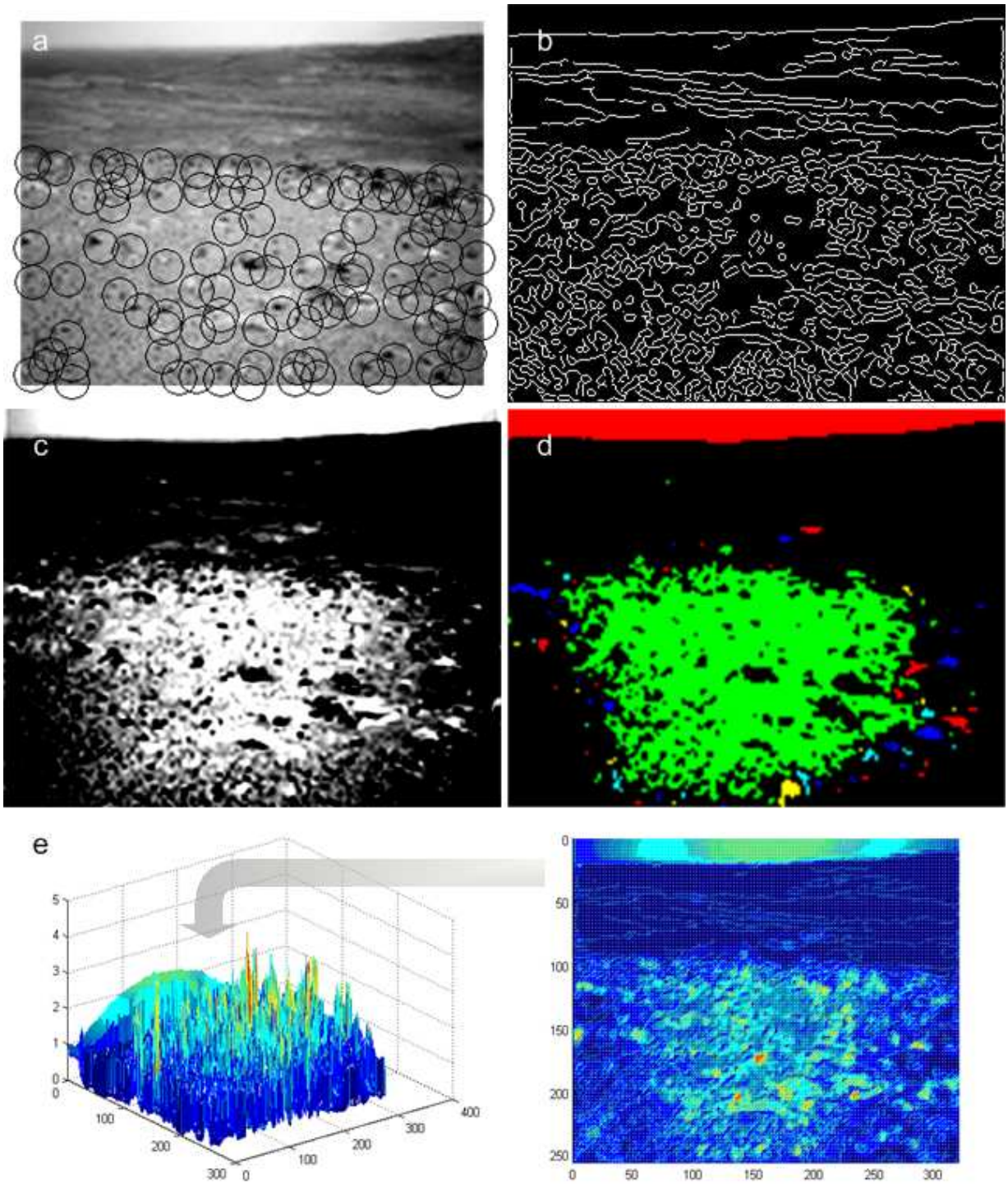


Figure 4.4: Mars terrain, taken by the left navigation camera of the Spirit Rover on its way to Gusev Crater [37]. Parts a-d display the output of various feature detectors: a) corner, b) edge, c) intensity contrast, d) connectivity (the different segments on the image show the different detected objects). Part e) shows the 3D mesh plot of the combined saliency map. The most salient point in this scene is located at pixel (153, 172), near the centre of the image.

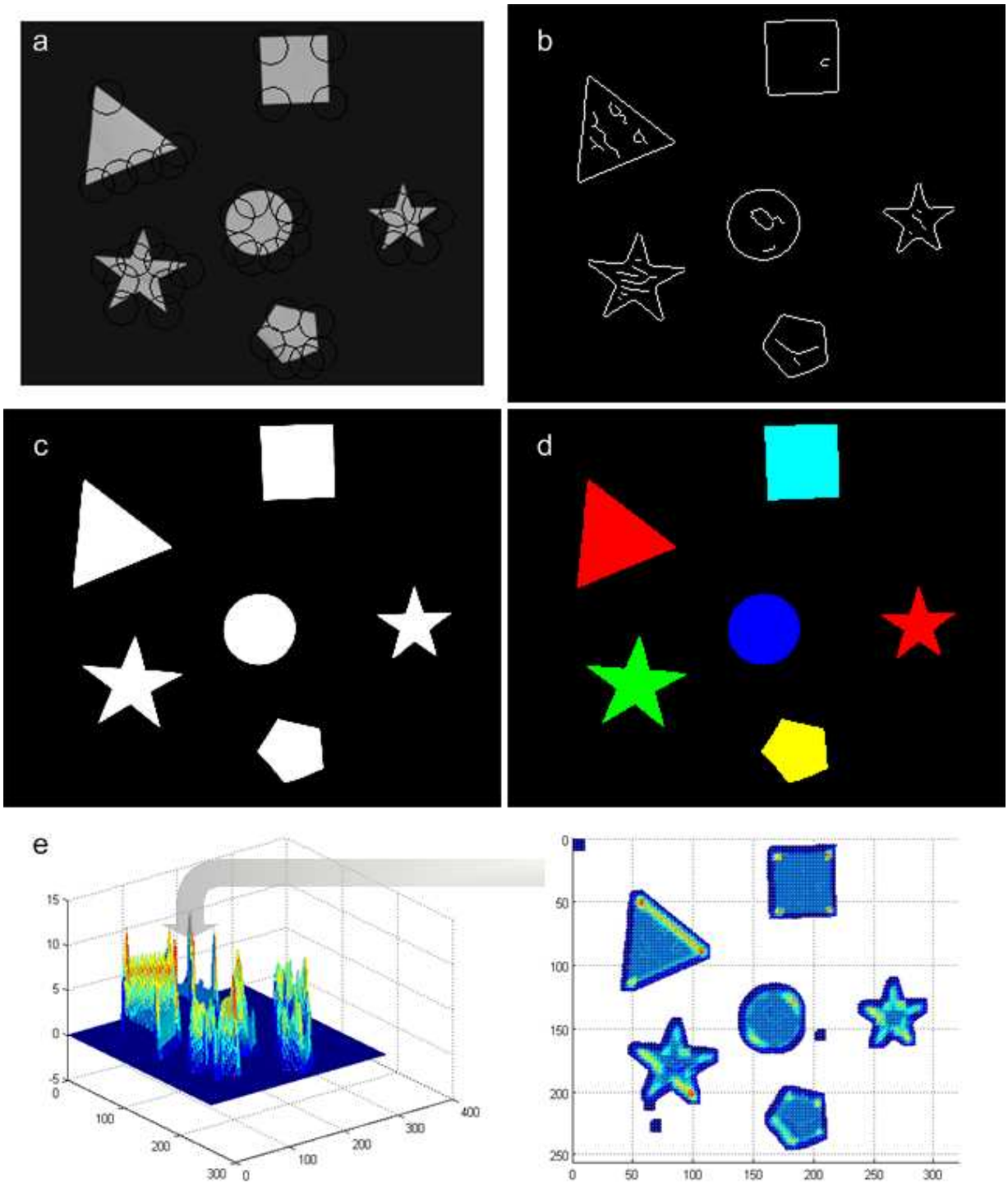


Figure 4.5: Test scene containing various geometric shapes. Parts a-d display the output of various feature detectors: a) corner, b) edge, c) intensity contrast, d) connectivity (the different segments on the image show the different detected objects). Part e) shows the 3D mesh plot of the combined saliency map. The most salient point in this scene is located at pixel (57, 50), the upper vertex of the triangle.

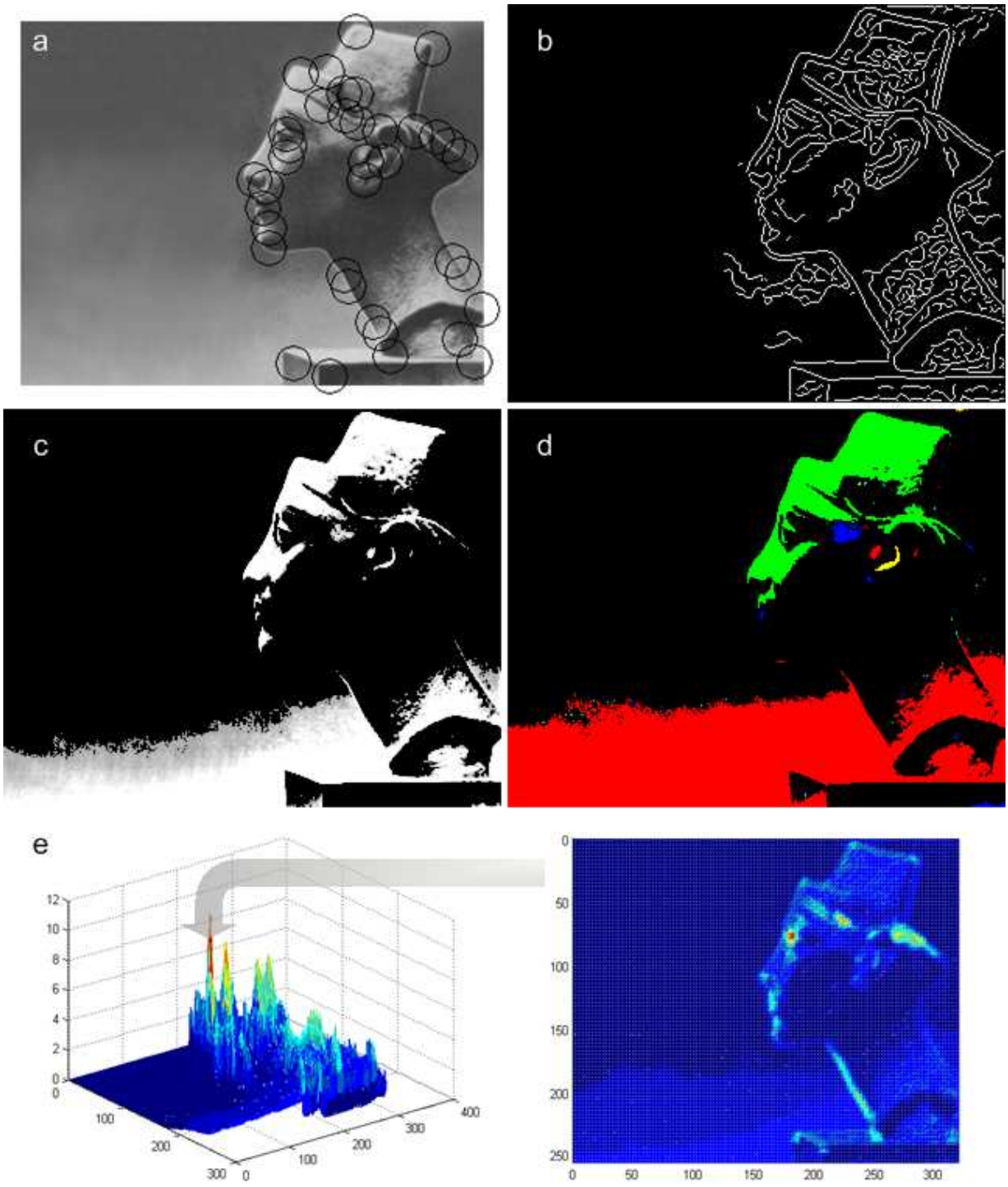


Figure 4.6: Bust of Nefertiti [3]. Parts a-d display the output of various feature detectors: a) corner, b) edge, c) intensity contrast, d) connectivity (the different segments on the image show the different detected objects). Part e) shows the 3D mesh plot of the combined saliency map. The most salient point in this scene is located at pixel (181, 77), Nefertiti's eye.

Figure 4.7 plots the 100 most salient points found in the test image processed in Figure 4.6. The points are connected in order of salience.

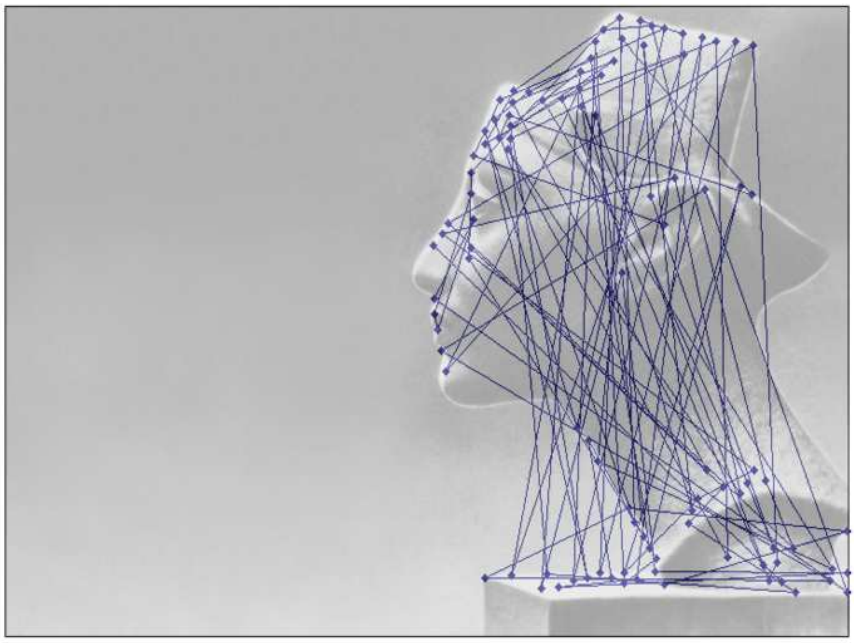


Figure 4.7: Top 100 salient points from the saliency analysis of the bust of Nefertiti.

The utility of the information provided by each of the feature detectors depends on the individual input image. In all of these test images, the corner detector seemed to provide the most amount of useful information. However, in Figure 4.1, the segmentation map is also quite useful. Similarly, intensity contrast would be useful in a scene with a homogenous background, such as in Figure 4.4, but with a distinctive object in the foreground, which would have a different texture and brightness than the background. The discussion will return to this topic at the end of the chapter. The following sections describe the methods by which features are extracted from raw images.

4.3 Feature Detector #1: Corners

Corners can either be detected by using a morphological hit and miss routine, where corner shapes are matched to the structuring element, or by finding the intersection of two sharp changes in intensity gradient. The main disadvantage of the hit and miss routine is that it will only find corners that are specifically described by the structuring elements. This can be improved by placing some don't care values in the structuring elements (to relax the definition of the corner shape); however a structuring element that is too general will result in the detection of false corners. The latter algorithm is much more complex, but also more robust and accurate; it was used to generate the results shown in Figures 4.1 - 4.6a.

4.3.1 Morphological Hit and Miss Operator

Morphology pertains to the shapes and structures of image objects; binary morphology operates on binary (black and white) images. The morphological hit and miss operator searches the image matrix for patterns that match the structure described by the kernel (structuring element). The hit and miss corner detector receives as input the binarized scene image and returns a second binary image: the pixels that are active in the output image are those whose neighbourhoods match the structure described by the structuring element. Each structuring element can only describe one corner shape/orientation; therefore, multiple passes of the hit and miss operator are necessary, using a variety of structuring elements that describe different corner orientations. This implementation uses four structuring elements to describe corners oriented at 0°, 90°, 180° and 270° from the positive x-axis.

A '1' in the structuring element indicates a pixel that is on (i.e. '1') in the input binary image, a '-1' indicates a pixel that is off (i.e. '0'), and a '0' indicates a don't care condition, where the output is true regardless of whether the input pixel is on or off. The don't care condition relaxes the corner shape specification; therefore the same structuring element can be used to detect sharp corners as well as rounded corners. The size of the kernel also determines the strictness on the corner shape; the larger the structuring element, the more pixels that need to be included in the corner shape. The smallest structuring element that describes a corner is a 3x3 matrix. It is tricky to determine optimal size of the structuring element. On the one hand, if the rules are too

$$topLeftMask = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 0 & 1 \\ -1 & -1 & 0 & 0 & 0 & 1 \\ -1 & -1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad topRightMask = \begin{bmatrix} 1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 0 & -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 & -1 & -1 \\ 1 & 0 & 0 & 0 & -1 & -1 \\ 1 & 1 & 0 & 0 & 0 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$botLeftMask = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & 0 & 0 & 0 & 1 & 1 \\ -1 & -1 & 0 & 0 & 0 & 1 \\ -1 & -1 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 \end{bmatrix} \quad botRightMask = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 & -1 \\ 1 & 0 & 0 & 0 & -1 & -1 \\ 1 & 0 & -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{4.1}$$

strict, then not many corners will be found; on the other, if the rules are too relaxed, then many false corners will be returned. Also, increasing the size of the structuring elements proportionally increases the number of iterations of the processing routine per pixel. In the set of structuring elements shown in Equation 4.1, the 6x6 matrices imposes the condition that each edge along the corner junction must contain at least 6 pixels, requiring 36 iterations per pixel.

The main advantages of the morphological hit and miss operator are its ease of implementation and its fast execution time (since only simple computations are required). However, there are several obvious disadvantages to this routine: i) only those corners whose acuteness and orientations match the shapes specifically described by the structuring elements will be found; and ii) the output decision is boolean (pass or a fail), and provides no information regarding the strength of a corner (therefore no degree of salience can be predicted through this routine). These disadvantages of the routine result in an expensive tradeoff between speed and accuracy, especially since the corner is generally the most useful feature to detect in the determination of an object's salience.

4.3.2 Detection of Intensity Gradient Changes

The gradient-based corner detection scheme essentially looks for the junction of two or more strong edges, which are defined as strong regions of intensity variation. The corner detection algorithm is given by [36]:

Inputs: 1. Image, I

2. Threshold parameter, τ_{corner}

3. Neighbourhood size parameter, N, where the window under consideration will be of size 2N+1

Output: 1. Co-ordinates of the most prominent corners in the scene

Algorithm: 1. Compute the spatial image derivatives along the x and y directions: $J_x = \partial I/\partial x, J_y = \partial I/\partial y$. These describe the gradient changes in intensity in the image, and are computed by convolving the columns (rows) of I with the kernel [1 0 -1].

2. For each point, p:

a. Let $J_{x,p}^2 = \partial I/\partial x$ and $J_{y,p}^2 = \partial I/\partial y$ in the (2N+1)x(2N+1) neighbourhood surrounding p. Compute the lower eigenvalue (if it exists), λ_2, of C, where:

$$C = \begin{bmatrix} \sum J_{x,p}^2 & \sum J_{x,p}J_{y,p} \\ \sum J_{x,p}J_{y,p} & \sum J_{y,p}^2 \end{bmatrix}$$

b. If $\lambda_2 > \tau_{corner}$, save the coordinates of p in the list L.

3. If there are multiple coordinates stored in L that belong to the same (2N+1)x(2N+1) neighbourhood, keep only the coordinates of the local maxima.

C describes the structure of the intensity changes in the (2N+1)x(2N+1) neighbourhood around a given point, p. Since C is a 2x2 matrix, it will have two eigenvalues: λ_1 and λ_2. These eigenvalues indicate the edge strengths in the local neighbourhood; the stronger the edges, the higher the eigenvalues. Therefore, if there are two strong edges with different orientations in the neighbourhood, then both λ_1 and λ_2 will be large. Mathematically, a corner exists if both eigenvalues are above the threshold, τ_{corner}. Since λ_1 is by definition larger than λ_2, only λ_2 2

needs to be computed and compared to τ_{corner}. λ_2 is given by[1]:

$$\lambda_2 = \frac{(\sum J_{x,p}^2 + \sum J_{y,p}^2) - \sqrt{4(\sum J_{x,p}J_{y,p})^2 + (\sum J_{x,p}^2 - \sum J_{y,p}^2)^2}}{2} \tag{4.2}$$

An appropriate value for the threshold, τ_{corner}, will vary for different images; rather than use a pre-defined value for τ_{corner}, the code (Appendix B) simply starts with a relatively low value of τ_{corner} to filter out most of the false corners, and then retains the top 100 prominent corners in the list.

The optimal value for N will also vary for different images. A choice of N between 2 and 10 will generally yield reasonable results [36]; in this implementation, N is chosen to be 4, such that the neighbourhood under consideration for corners is 9x9. This neighbourhood size was determined experimentally using a set of test images.

Figure 4.8 shows a plot of the corners detected by a) the gradient-based corner detection algorithm (diamonds), b) a hit-and-miss operation using a set of four 5x5 kernels (squares), and c) a hit-and-miss operation using a set of four 6x6 kernels (triangles). The output of the hit-and-miss using the 5x5 kernels present closer results to the gradient-based corner detector than the hit-and-miss operation using the 6x6 kernels. However, the 5x5 kernel operation resulted in 203 detected corners; in the gradient-based method, the code is instructed to select the peak 100 detected corners. Since there is no way to differentiate between the strengths of the corners detected by the hit-and-miss operator, it is necessary to accept the entire output set as valid. The only way to reduce the number of corners that are detected are to change the structuring elements and tighten their corner descriptions. However, the results from the hit-and-miss operator using the set of 6x6 kernels did not produce much better results. In fact, if the diamonds are used as a measure of corner validity, then results of the hit-and-miss operator (using both kernel sizes) are incomplete. Unfortunately, there is no way to control how many corners are returned by the hit-and-miss operator. Furthermore, the corners that do not coincide with the four orientations described by the structuring elements are left undetected.

[1]Note: The larger eigenvalue, λ_1, is the addition, rather than the subtraction, of the two major terms.

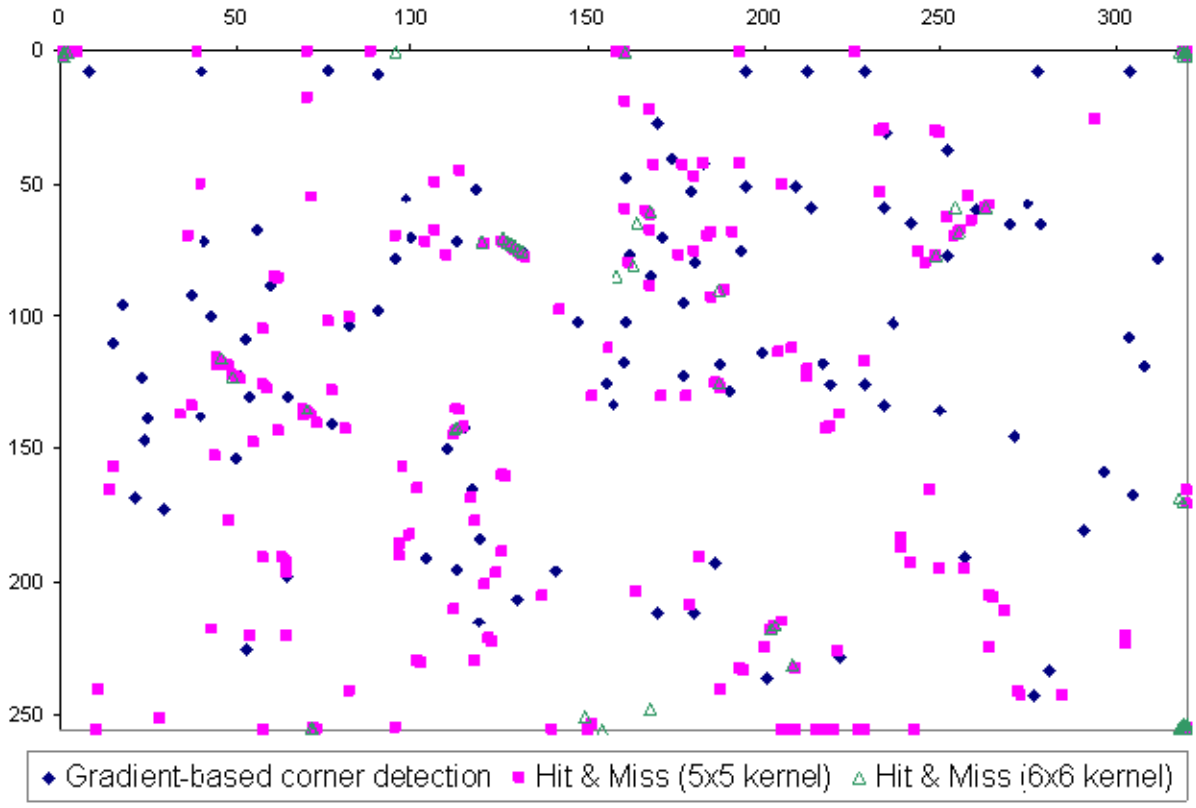
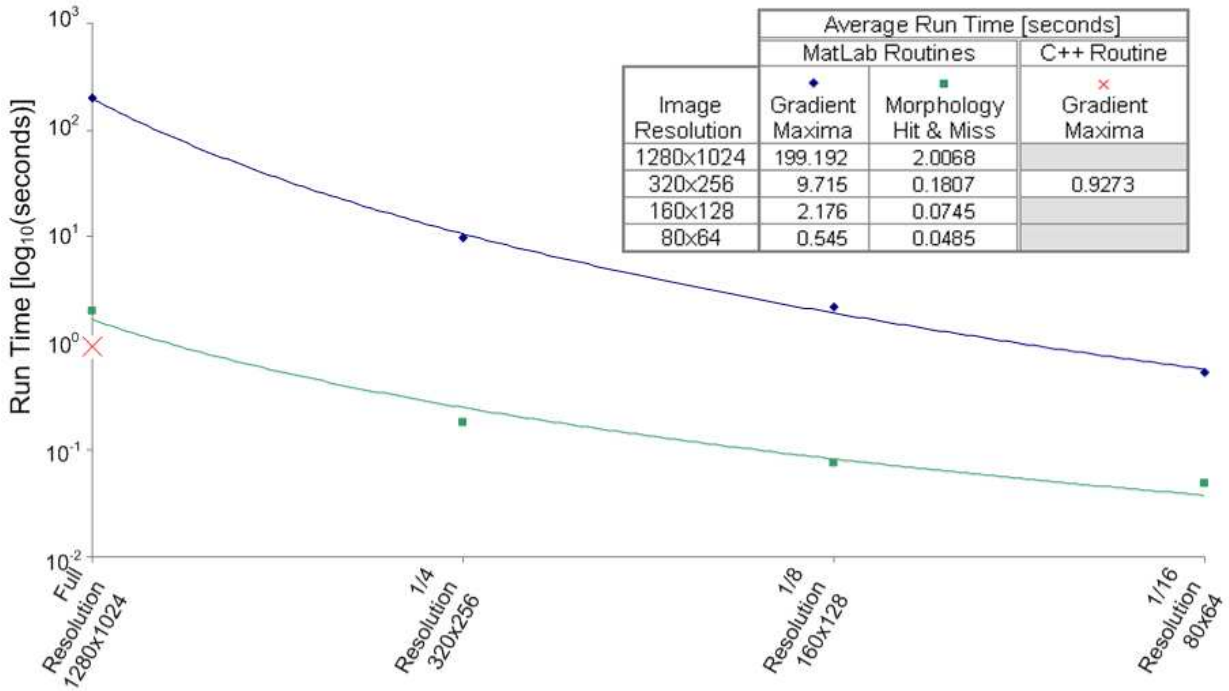


Figure 4.8: Comparison of results from the two corner detection algorithms. Diamonds denote results from the gradient-based corner detection algorithm, squares denote results from a hit-and-miss operation using a set of four 5x5 kernels (squares), and triangles denote a hit-and-miss operation using a set of four 6x6 kernels.

4.3.3 Runtime Comparison

Figure 4.9 plots the average execution time to run the gradient-based and hit-and-miss algorithms. These execution times are averaged over 100 samples, using a variety of input images. For comparison, the run-time of the gradient-based routine was also measured in the C++ prototype implementation. Both executions were run on the same computer, using a 2.4GHz processor and 256MB memory. For the analysis of a 1/4 SXGA image, the hit-and-miss operator performed over 50 times faster than the gradient-based algorithm, implemented in MatLab. However, while the hit-and-miss operator (which is a built-in function provided in MatLab's Image Processing ToolKit [38]) is an optimized routine, the gradient-based corner detector is a non-optimized routine written by the thesis author; it was implemented for the intention of testing the function of the algorithm, rather than the optimization of its performance. Fortunately, the same algorithm implemented in C++, which is used in the Mike prototype, ran an order of magnitude faster than the MatLab code.



	Average Run Time [seconds]		
	MatLab Routines		C++ Routine
Image Resolution	◆ Gradient Maxima	■ Morphology Hit & Miss	× Gradient Maxima
1280x1024	199.192	2.0068	
320x256	9.715	0.1807	0.9273
160x128	2.176	0.0745	
80x64	0.545	0.0485	

Figure 4.9: Runtime of corner detection algorithms.

4.4 Feature Detector #2: Edges

Edges, like corners, can also be detected through a combination of morphological operations, or through the detection of gradient changes. There are many gradient-based methods of detecting image edges, including the Sobel, Roberts, Prewitt, and Canny edge detectors. The following sections discuss the morphological and Canny edge detectors.

4.4.1 Morphology

Before the morphological edge detector can be understood, it is necessary to first understand the binarization process.

Binarization

Binarization is the process of converting greyscale images to a binary image: an image composed of pixels that hold one of two values, '1' (foreground) or '0' (background). A pixel is '1' if the original image's grey value is above the binarization threshold; otherwise it is '0'.

If a good binarization threshold can be found, morphology can yield comparable results to its much more complex counterparts (i.e. Sobel and Canny). Unfortunately, the determination of a proper binarization threshold can prove to be a complex task in itself. Binarization is achieved differently in the C++ and MatLab implementations (see Appendix A and B). In the C++ prototype code, the binarization threshold is set to two standard deviations above the mean grey value. This was determined experimentally to provide adequate binarization results, with a tradeoff between complexity (speed) and quality (distinction between foreground and background). The MatLab code uses the built-in Otsu routine to find a proper threshold. Otsu's method determines an image-specific binarization threshold level that mimimizes the intraclass variance in the image. Although Otsu's method is generally agreed to be a good method for determining binarization thresholds [39],[40], it can still often remove important detail from foreground objects. No known binarization technique provides consistently perfect separation between foreground and background for all images. Figure 4.10 shows binary images obtained using various thresholding methods.

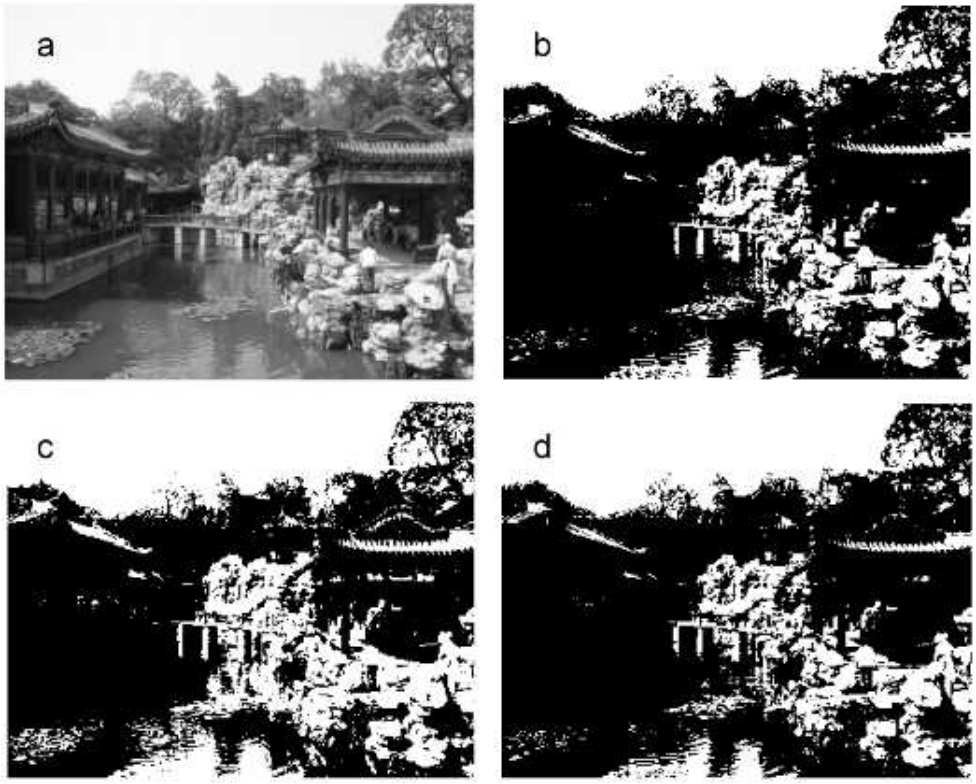


Figure 4.10: Binarization using various threshold levels. a) Original 320x256 greyscale image. b) Binary image thresholded by a value calculated by Otsu's method. c) Binary image thresholded by the mean grey value. d) Binary image thresholded by two standard deviations above the mean grey value.

Morphological Edges

Edges are determined by finding the outlines of shapes. There are two main morphological operators: erosion and dilation. Erosion removes a layer of foreground pixels along the perimeter of an object; it effectively shrinks objects and opens holes within an object. Dilation adds a layer of foreground pixels along the perimeter of an object; it effectively enlarges objects and closes small holes. Figure 4.11 illustrates this process. Conceptually, erosion can be considered as an 'AND' operation, while dilation can be considered as an 'OR' operation. For erosion, if all of the pixels surrounding an active foreground pixel are also active, then the corresponding result is true; otherwise it is false. For dilation, if any of the pixels surrounding the pixel under consideration are active or if the pixel itself is active, then the result is true; otherwise it is false. Object outlines can have two definitions:

1. Background pixels lying on the boundary of the object; or
2. Foreground pixels along the perimeter of the object.

The first type of outline is achieved by subtracting the original binary image from its dilated image. Similarly, the second type of outline is achieved by subtracting the eroded image from the original binary image. The definition of background and foreground is grossly generalized: the binarization process assumes that the background is darker than the foreground objects. This is not always the case. However, both methods of determining outlines will find the border between distinctively light and dark areas in the scene.

4.4.2 Canny Edge Detector

Given the speed and ease of implementation for a morphological edge detector, the results of morphology are good enough for this application (since the scan mode only requires pre-processing to roughly determine salient regions and edge information is the lowest weighted feature). However the gradient-based corner detection scheme is very similar to Canny edge detection, and many of the intermediate values that are computed during corner detection can be reused in the Canny calculation. Therefore, although the morphological routine is faster, an implementation of the

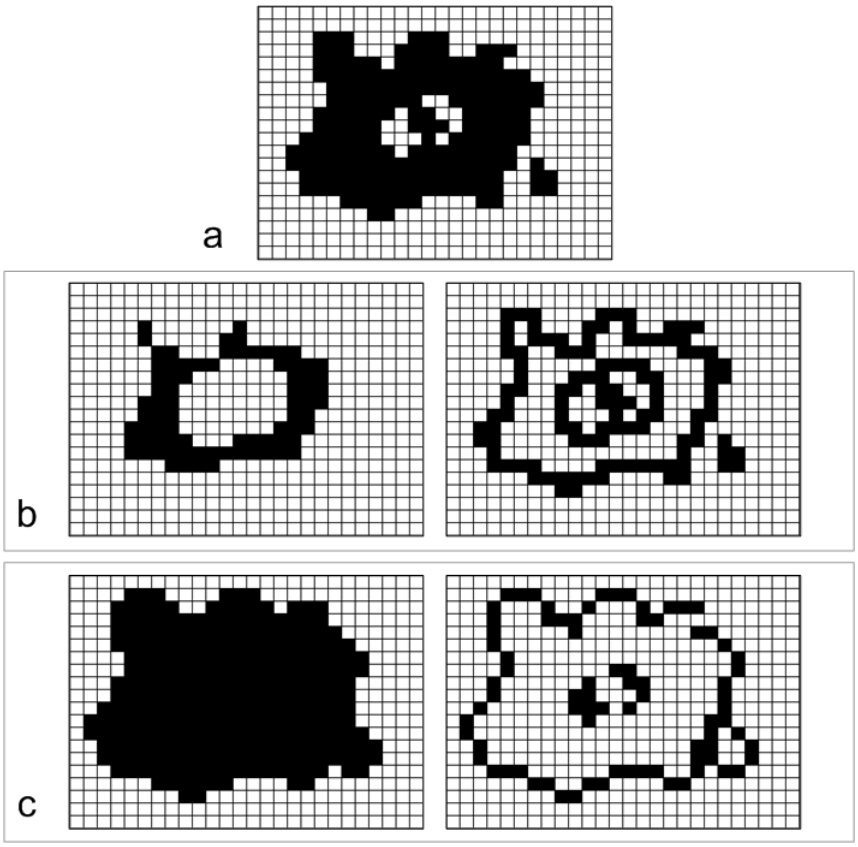


Figure 4.11: Effects of erosion and dilation. a) Original binary image. b) Left: Eroded image, right: original image minus eroded image. c) Left: Dilated image, right: dilated image minus original image.

Canny edge detector is also available in the Mike C++ code (Appendix A). The Canny edge detector essentially looks for local maxima of intensity variation. Its algorithm is given by [36]:

Input: Image, I

Output: Boolean (binary) image depicting detected edges, I_{edge}

Algorithm: 1. Convolve I with a Gaussian matrix (a matrix of quantized Gaussian values) to smooth the image and filter noise. This is called Gaussian smoothing.

2. Compute the spatial image derivatives along the x and y directions: $J_x = \partial I/\partial x, J_y = \partial I/\partial y$. These describe the gradient changes in intensity in the image, and are computed by convolving the columns (rows) of I with the kernel [1 0 -1]. (Note: this is the same as step 1 in gradient-based corner detection).

3. Compute the edge strength matrix, Es, where each element,

$$e_s(i,j) = \sqrt{J_x^2(i,j) + J_y^2(i,j)}$$

Edge strengths in Es are distributed in the surrounding neighbourhood. The true edges are located at the local peaks along an edge orientation.

4. Compute the edge orientation matrix, Eo, where each element,

$$e_o(i,j) = arctan\frac{J_y}{J_x}$$

5. Thin the edges described by Es to determine the local maxima and store the result in the matrix I_{edge}. This is called Canny suppression. In this step, four edge directions are considered:
$d_1 = 0°$, $d_2 = 45°$, $d_3 = 90°$, and $d_4 = 135°$.

a. Determine the direction d_k that best describes the edge orientation at each $E_o(i,j)$.

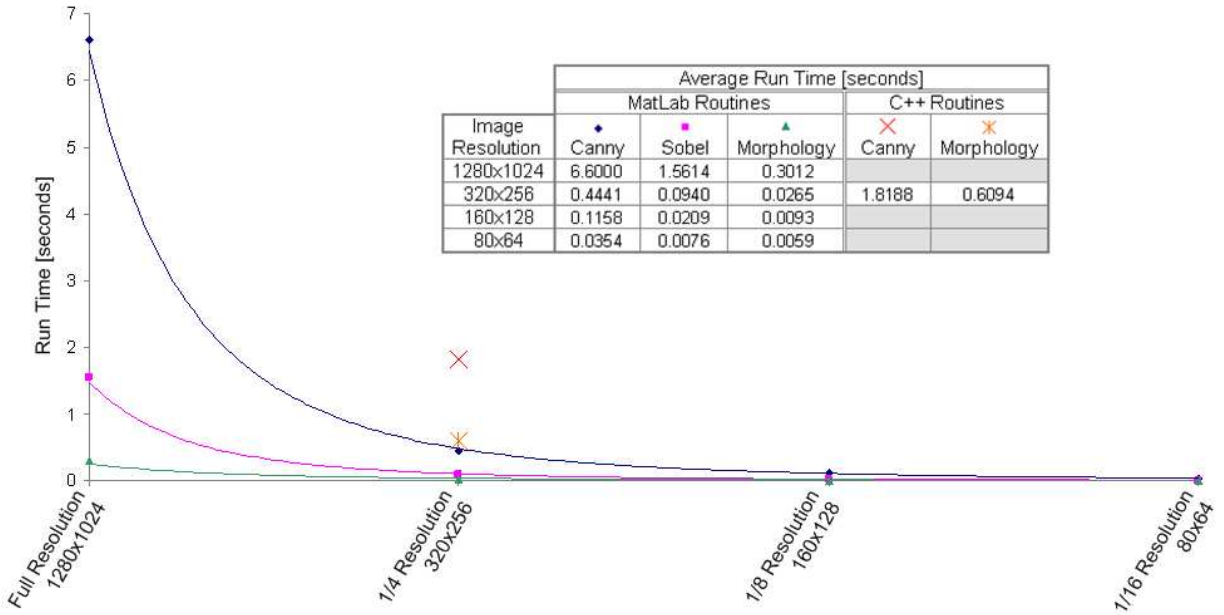
b. Compare $E_s(i,j)$, with the edge strength of its two neighbours along the direction d_k. If $E_s(i,j)$ is not the maximum, then the pixel at (i,j) does not belong to an edge. $I_{edge}(i,j)$ is assigned the value of $E_s(i,j)$ if it is a maximum, 0 otherwise.

A final step in the Canny algorithm is hysteresis thresholding, where the edge strengths are compared with a minimum threshold level. This filters out all weak edges and it improves

the quality of results from the Canny detector. However, this added level of complexity is not necessary for the construction of the saliency map, which is only intended to be a rough sketch. Thus this final step is skipped in the prototype implementation.

4.4.3 Runtime Comparison

Figure 4.12 plots the average execution times to run the two edge detection algorithms described in the previous sections. The execution times were averaged over 100 samples, using a variety of input images to test the performance of the algorithms. In the C++ implementation, morphology took 2/3 less time to run than the Canny edge descriptor. However, it should be noted that the Canny routine shares some computations with the corner detection scheme, and therefore the combined runtime between corner detection and Canny edge detection is less than their individual totals. Still, morphological edge detection was much faster and is chosen to be the active routine in the prototype code.



Average Run Time [seconds]					
	MatLab Routines			C++ Routines	
Image Resolution	Canny	Sobel	Morphology	Canny	Morphology
1280x1024	6.6000	1.5614	0.3012		
320x256	0.4441	0.0940	0.0265	1.8188	0.6094
160x128	0.1158	0.0209	0.0093		
80x64	0.0354	0.0076	0.0059		

Figure 4.12: Runtime of edge detection algorithms.

The quality of results from the morphological edge detector is highly dependent on the quality of results from the binarization process. Although both Otsu's method and the method used in the prototype code provide reasonable results, they still do not isolate all objects properly in a complex scene. This method will find the outlines of most of the major shapes in the scene. Since edges are given a low priority in the salience sequence, the inaccuracy of the edge detector is not a major concern. However, in an application where edge information is important and the edge is given a higher weighting, then perhaps the Canny detector should be used to gain more accurate results.

4.5 Feature Detector #3: Intensity Contrast

The purpose of the intensity contrast routine is to locate any abnormally bright or dark objects in the image. It calculates the absolute difference between a pixel's intensity and the mean grey value of the overall image, and returns the pixel's deviation from the mean if it is more than a predefined threshold value:

$$I_{contrast}(i,j) = \begin{cases} |I(i,j) - MeanVal|, & if|I(i,j) - MeanVal| \geq \tau_{contrast} \\ 0, & otherwise \end{cases} \tag{4.3}$$

$\tau_{contrast}$ is a parameter that determines how much brighter/darker an object needs to be in relation to the background, in order to be considered salient.

4.6 Feature Detector #4: Connectivity

The connectivity between salient points is loosely determined by grouping pixels that belong to the same object. For a static scene, the intention of this routine is to provide a deciding factor between two similarly strong corners that belong to different objects. The high-priority objects and their features will therefore be attended first, which is important in case there is an interruption to the saccading process (such as the sudden detection of motion). For a dynamic scene (where one or more objects are moving), it is intended for to aid in tracking objects.

The implementation of connectivity is achieved differently in the MatLab test code and the

C++ prototype code. In the MatLab test code, the image is first converted to a binary image and the built-n MatLab bwlabel routine [38] groups the foreground objects. The bwlabel routine assigns a numerical label to all the segmented objects in the image; all pixels belonging to the same object share the same label.

In the C++ prototype code, the connectivity implementation categorizes all the corners, rather than all the foreground pixels, in the image. The membership of two corners to the same object is defined by an imaginary line drawn between those two corners. If the line crosses over only foreground pixels, then the two corners are assigned the same object label. Frame 1 of Figure 4.13 shows four distinct shapes with the corners grouped together. The lines in Figure 4.13 have been *superimposed* onto the image to indicate the corners that have been grouped together (i.e. determined to belong to the same object).

Although this connectivity definition works well for this type of input image, where shapes are simple and distinct, it has not been tested on more complex images. The ability to track the corners of an object, however, is potentially useful in tracking moving targets, especially when the moving target moves in front of or behind other foreground objects and the two or more objects become indistinguishable during the target motion.

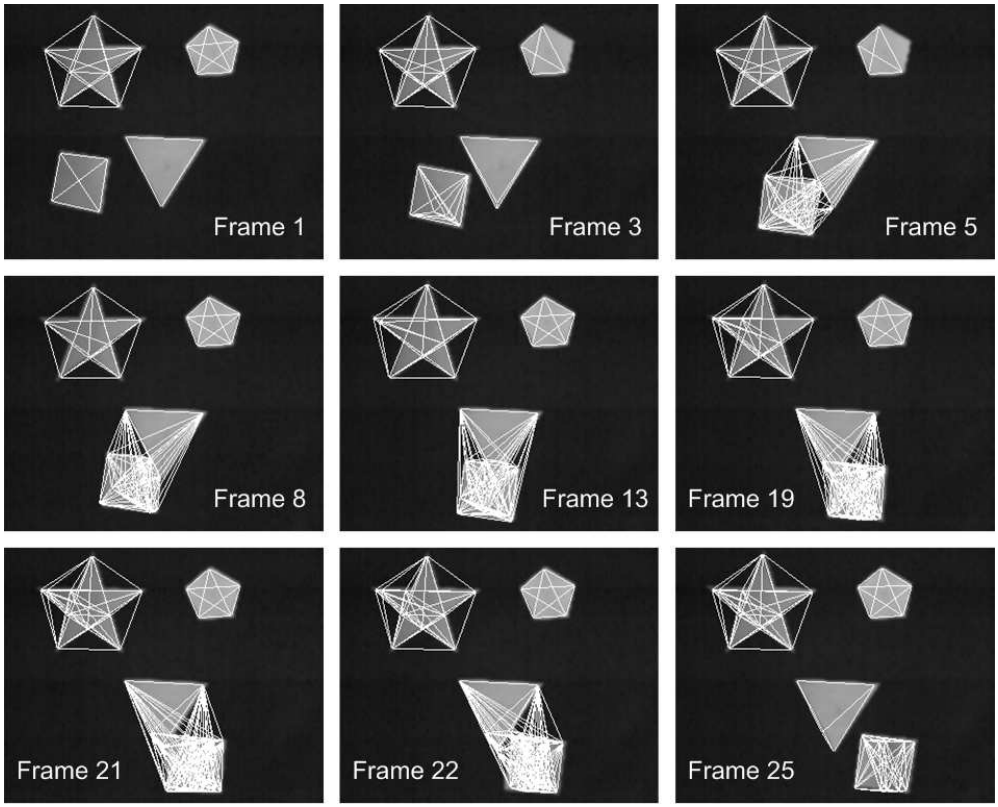


Figure 4.13: Image features grouped by their connectivity. Lines are drawn between each corner that belongs to the same object.

4.7 Top-Down Visual Search

The previous discussion considered the construction of the saliency map based on a bottom-up visual search. This section briefly considers the conditions on salience for a top-down visual search.

A top-down visual is task-oriented, and therefore requires a basic understanding of the features, or combinations of features, that categorize objects to judge their relevance to the task.

4.7.1 Learning and Classification

In 1961, Shepard conducted a series of experiments to determine the complexity of classifying objects based on a set of stimuli. He defined classification as "a grouping of a given set of stimuli into two or more mutually exclusive and exhaustive classes [41]." Using three sets of binary stimuli (i.e. having two possible values), Shepard investigated six different ways of classifying eight objects based on the stimuli: colour (grey or white), size (large or small), and shape (square or triangle). Although there are 70 ($\frac{8!}{4!^2}$) possible combinations of 2-group arrangements of the eight objects, there are only six unique types of classifications; the rest are all variants of these six types [41]. Figure 4.14 shows examples of these six types of classifications discussed by Shepard.

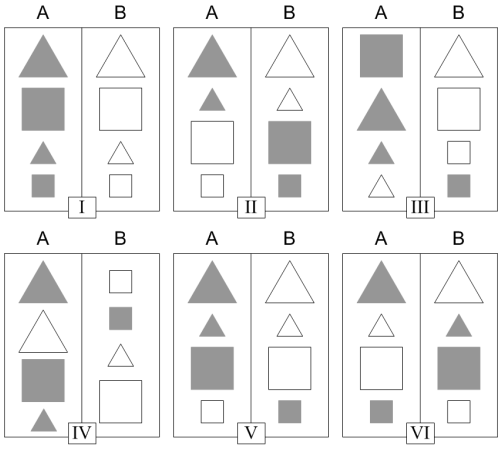


Figure 4.14: Examples of the six types of categorizing objects based on three binary features.

Type I : The "necessary and sufficient condition [41]" for the objects in column A is that each object must be grey. Similarly, the objects in column B must be white. Therefore, only the colour feature is considered; size and shape are ignored. *Type I classification is based on the value of one feature.*

Type II : The objects in column A are either i) grey and triangular, or ii) white and square. *Type II classification is based on the combination of values from two features.*

Types III-VI *: Types III-VI classifications require consideration of values from all three features; however, they differ in the logical evaluation of the features.*

Type III : The objects in column A are either i) grey and large, or ii) small and triangular. Thus, although all three features must ultimately be considered, each condition only requires the consideration of two features.

Type IV : The objects in column A are either i) large and triangular, or ii) large and grey, or iii) grey and triangular. Here, there are three conditions, but two features considered in each condition.

Type V : The objects in column A are either i) grey and triangular, or ii) large and grey, or iii) square and small and white. Two of the three conditions require the evaluation of two features; the third condition evaluates all three features.

Type VI : The objects in column A are either i) grey and large and triangular, or ii) small and white and triangular, or iii) grey and small and square, or iv) white and large and square. The four separate conditions each evaluate all three features.

According to Shepard, the complexity of classification is ranked in the following order: Type I $<$ II $<$ (III, IV, V) $<$ VI [41]. This ordering of classification complexity suggests that the process of learning how to categorize objects requires that the subject also learns how to selectively attend only to the features specifically relevant for a given type of classification [42]. Evidence supporting this hypothesis was presented by Rehder, who measured the eye movements of subjects presented

with classification tasks of eight objects that differed in three binary features [43]. Rehder found that, after an initial period of learning, subjects attended to only those features relevant to the rules of the classification type, and ignored the features that were not specified by the rules [43]. Based on these findings, Zhang developed a category learning task model that selectively attends to relevant features and develops a fixation pattern for each of the six classification types [44].

4.7.2 Categorical Learning in Mike

With the addition of a set of rules describing Shepard's the six classification types [41], Mike can be trained to perform basic categorical learning using the output of its feature detectors. Using the example shown in Figure 4.14, the following rules describe the classification of objects according to the six types.

Assigning a boolean descriptor for each feature, let:

isGrey	=	1	:	grey;	isGrey	=	0	:	white
isLarge	=	1	:	large;	isLarge	=	0	:	small
isSquare	=	1	:	square;	isSquare	=	0	:	triangle

An object is in column A if:

$$
\begin{aligned}
&TypeI: && isGrey = true \\
&TypeII: && (isGrey \otimes \overline{isSquare}) \oplus (\overline{isGrey} \otimes isSquare) \\
&TypeIII: && (isGrey \otimes isLarge) \oplus (\overline{isLarge} \otimes \overline{isSquare}) \\
&TypeIV: && (isLarge \otimes \overline{isSquare}) \oplus (isGrey \otimes isLarge) \oplus (isGrey \otimes \overline{isSquare}) \\
&TypeV: && (isGrey \otimes \overline{isSquare}) \oplus (isGrey \otimes isLarge) \oplus (\overline{isGrey} \otimes \overline{isLarge} \otimes isSquare) \\
&TypeVI: && (isGrey \otimes isLarge \otimes \overline{isSquare}) \oplus (\overline{isGrey} \otimes \overline{isLarge} \otimes \overline{isSquare}) \\
& && \oplus (isGrey \otimes \overline{isLarge} \otimes isSquare) \oplus (\overline{isGrey} \otimes isLarge \otimes isSquare) isSquare)
\end{aligned}
\tag{4.4}
$$

A routine containing the above rules incorporated with some minor modifications to Mike's feature detectors (Appendix B), analyzes the objects in the input test image shown in Figure 4.15. The three ternary variables, isGrey, isLarge and isSquare, can hold a possible value of 'true', 'false', or 'uncertain.' The variables are initialized to the 'uncertain' state. For each

object in the image, the isGrey flag is evaluated from the output of the intensity contrast and connectivity routines; it is 'true' if the object is grey (i.e. having an intensity value between 100-200), 'false' if the object is white (i.e. having an intensity value above 200), and 'uncertain' if the object contains both white and grey pixels or if the object's grey level is less than 100. The isLarge flag can be evaluated using the connectivity routine; it is 'true' if the bounding box around the object contains more than 60 pixels along its diagonal, 'false' if the diagonal is between 10-60 pixels, and 'uncertain' if the diagonal is less than 10 pixels. The isSquare flag is evaluated based on the output of the corner detection and connectivity routines; it is 'true' if the object contains four vertices, 'false' if it contains three vertices, and 'uncertain' if the object contains any other number of vertices. Figure 4.16 shows the output of the classification routine. Figure 4.17 shows the classification results if corner detection is disabled, and Figure 4.18 shows the classification results if intensity contrast is disabled. The question marks indicate that there is insufficient information from the detected features to classify the objects based on the rules in Equation 4.4. It is apparent that Type I classification can still be possible if some detectors are turned off, but Types III-VI, which are more complex, rely on the output of all the detectors. Although Type II classification only requires two out of the three feature dimensions described by Shepard [41], the size of the object is calculated by Mike's connectivity routine; the connectivity routine is therefore essential for all types of classification as it is used to segment the objects.

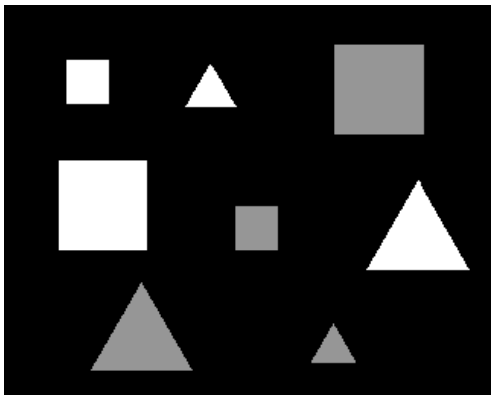


Figure 4.15: Input image for object cateogorization.

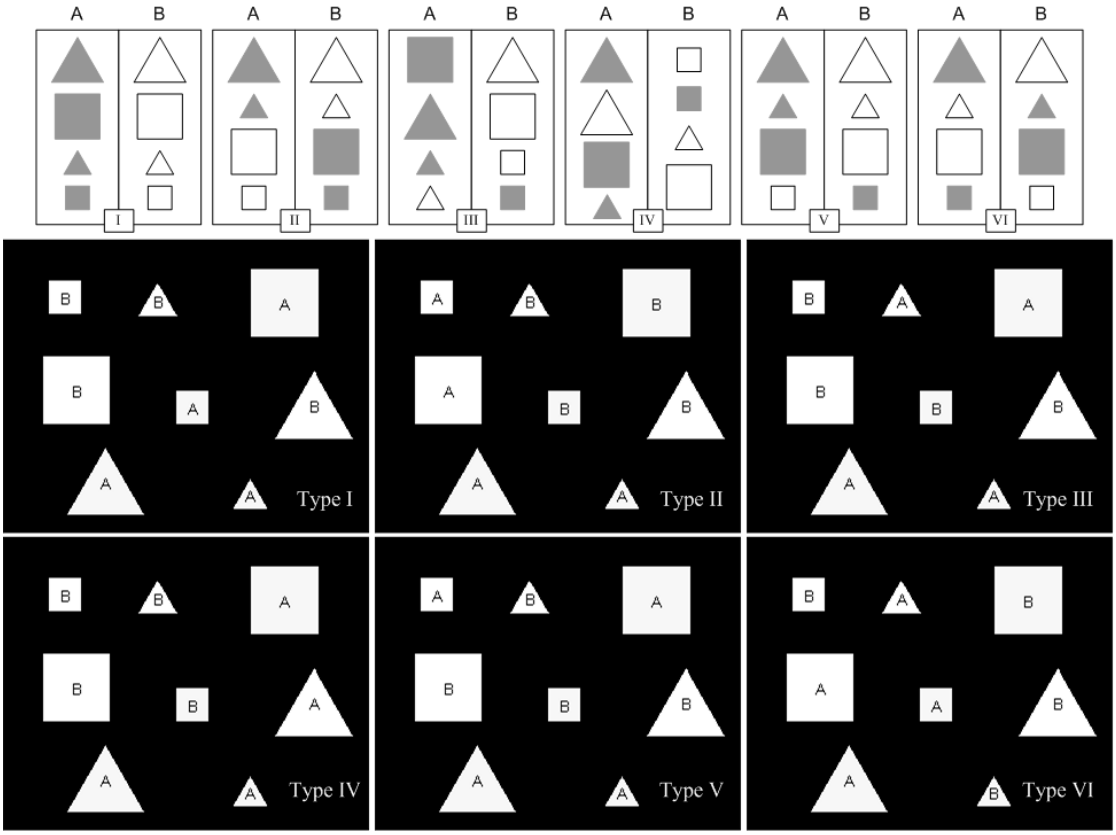


Figure 4.16: Classification of objects using Mike's feature detectors.

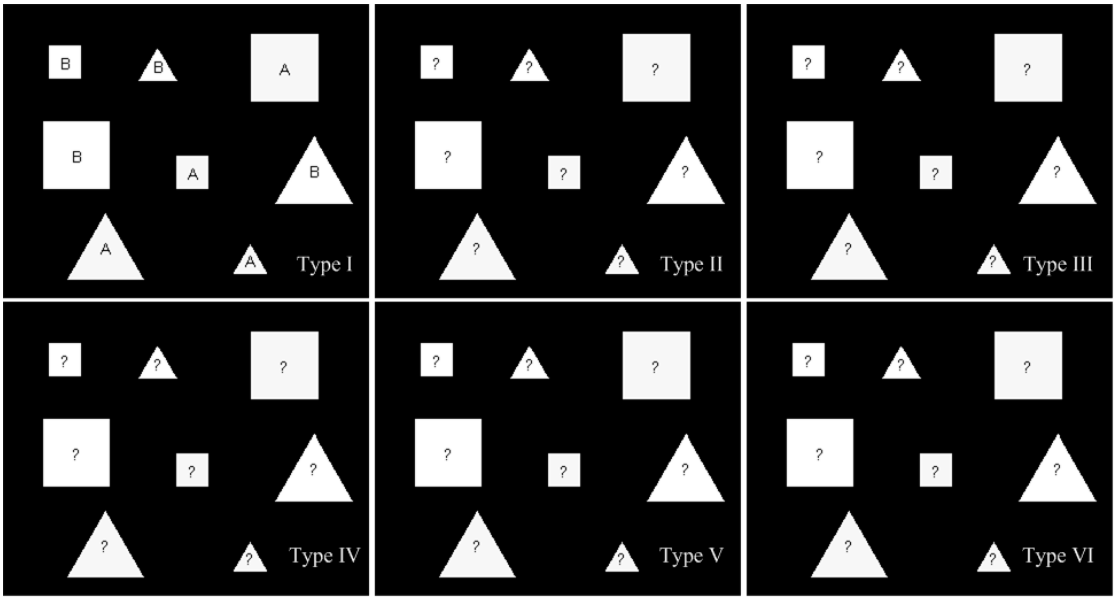


Figure 4.17: Object classification with corner detector disabled.

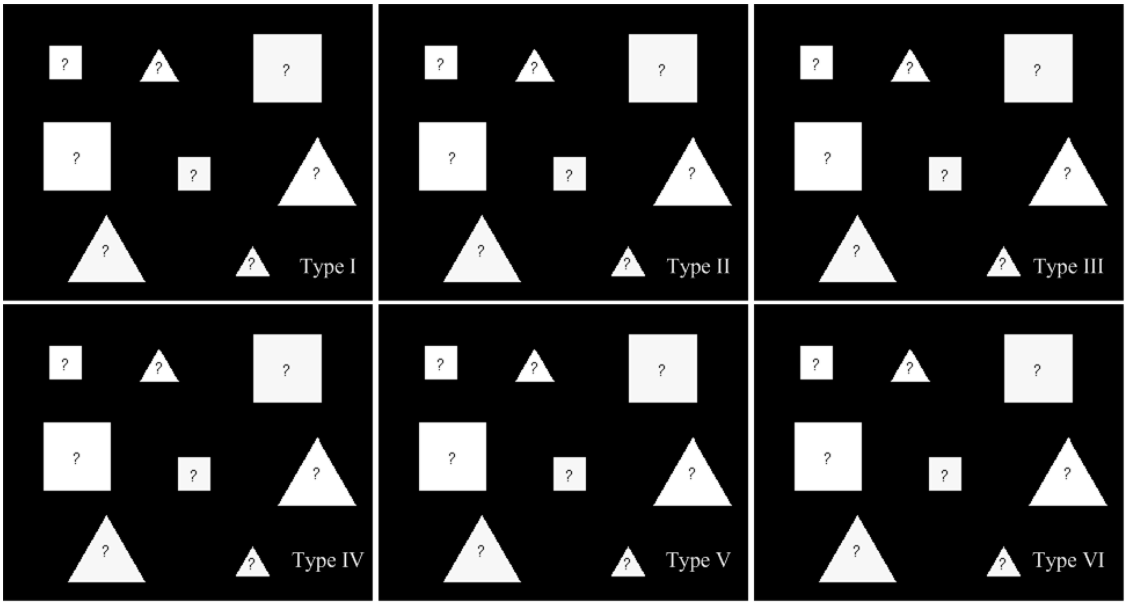


Figure 4.18: Object classification with intensity detector disabled.

4.7.3 Weighing the Features

The saliency map is a weighting function. Its output depends on the detected features, F_i, and their respective assigned weights, k_i. For n feature detectors, the salience at a point is given by:

$$Salience(x, y) = \sum_{i=0}^{n-1} k_i F_i(x, y) \tag{4.5}$$

Certain combinations of features and weights would be appropriate for some types of applications, while different combinations would be appropriate for other applications. If each of Mike's four feature detectors is assigned a unique ranking, then there are 24 (4!) possible combinations of rankings. However, the feature detectors are not required to be assigned unique rankings; two or more features can have the same ranking. Moreover, one or more features may be disabled, and the weight values, k_i, can be any natural number. Therefore, there exists an infinite number weight assignments for the feature detectors. Rather than assign arbitrary values of k to each feature, there needs to be a systematic way to determine the appropriate weight assignments of feature detection for a given application.

The saliency maps from Figures 4.1-4.6 were generated by assigning a weight of 7 to the corner detector, 5 to the intensity contrast calculation, 3 to the edge detector, and 1 to the connectivity indicator. This set of weightings was determined through experimentation, and worked well for the test images. Although these weightings can be used for general bottom-up searches, specific applications might require an emphasis on certain types of features. Tables 4.1-4.4 qualitatively demonstrate the effects on the saliency calculation (Equation 4.5) as emphasis is varied for Mike's four feature detectors on the images shown in Figure 4.19. These images represent sample data that might be used in autonomous navigation applications, security/biometric applications, and industrial inspection applications. Tables 4.1-4.4 also show the salient areas that would be studied during different durations. In his experiments, Yarbus noted that when a subject initially looks at a scene, the subject will first attend to the features that are most important to the scene's interpretation. When more time is allotted for study, the subject will then move on to some of the secondary regions of the scene, and perhaps also return to the top salient areas to fixate on them for longer [3]. Assuming an average of 3 saccades per second [18], Tables 4.1-4.4 show

the corresponding fixation points for examination durations of 5 seconds (15 salient points), 20 seconds (60 salient points), 45 seconds (135 salient points), and 70 seconds (210 salient points).

Tables 4.1-4.4 indicate that the strongest corners in the image dominate the top salient points, even when the corner detector is assigned a low weighting. Since corners mark the junction between two strong edges, the edge detector will also indicate salience in a corner region. Furthermore, the junction of strong edges indicates a sharp change in intensity; there is typically a large contrast value in that region, thus the intensity contrast calculation will also indicate salience in the neighbourhood of a strong corner. However, after the initial set of strong corners are exhausted, the edge detector, intensity contrast calculator, and connectivity indicator do not provide sufficient information to differentiate the salience between the secondary regions. This is apparent in the bottom row of images in Tables 4.1-4.4, where the corner detector is disabled; the remaining three detectors do not provide many new salient points after the initial 15 points.

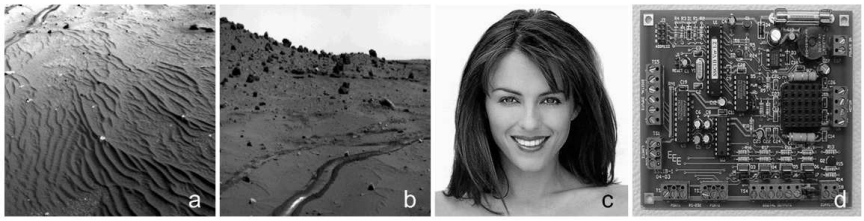


Figure 4.19: Sample images for autonomous navigation, security and industrial inspection applications. a) and b) Images of Mars terrain taken from the navigation camera on Spirit Rover [37]. c) Actress and model Elizabeth Hurley [45]. d) Circuit board [46].

For general applications, it is recommended that the corner detector be enabled and assigned a strong weighting relative to the other feature detectors. Unless the specific application requires object outline information, the edge detection may be redundant after corner detection, as corner detection will generally provide hits on most strong edges. It should be noted however, that corner detection will not detect edges do not intersect with other edges, such as along a surface horizon. The intensity contrast calculation can be useful in distinguishing unusually bright or unusually dark objects relative to the rest of the scene. Although the connectivity indicator is useful for grouping salient points that belong to the same object, its results are generally meaningless on their own. Therefore, the connectivity indicator should be assigned a relatively low weighting.

Table 4.1: Duration of scene examination versus feature weighting, autonomous navigation application.

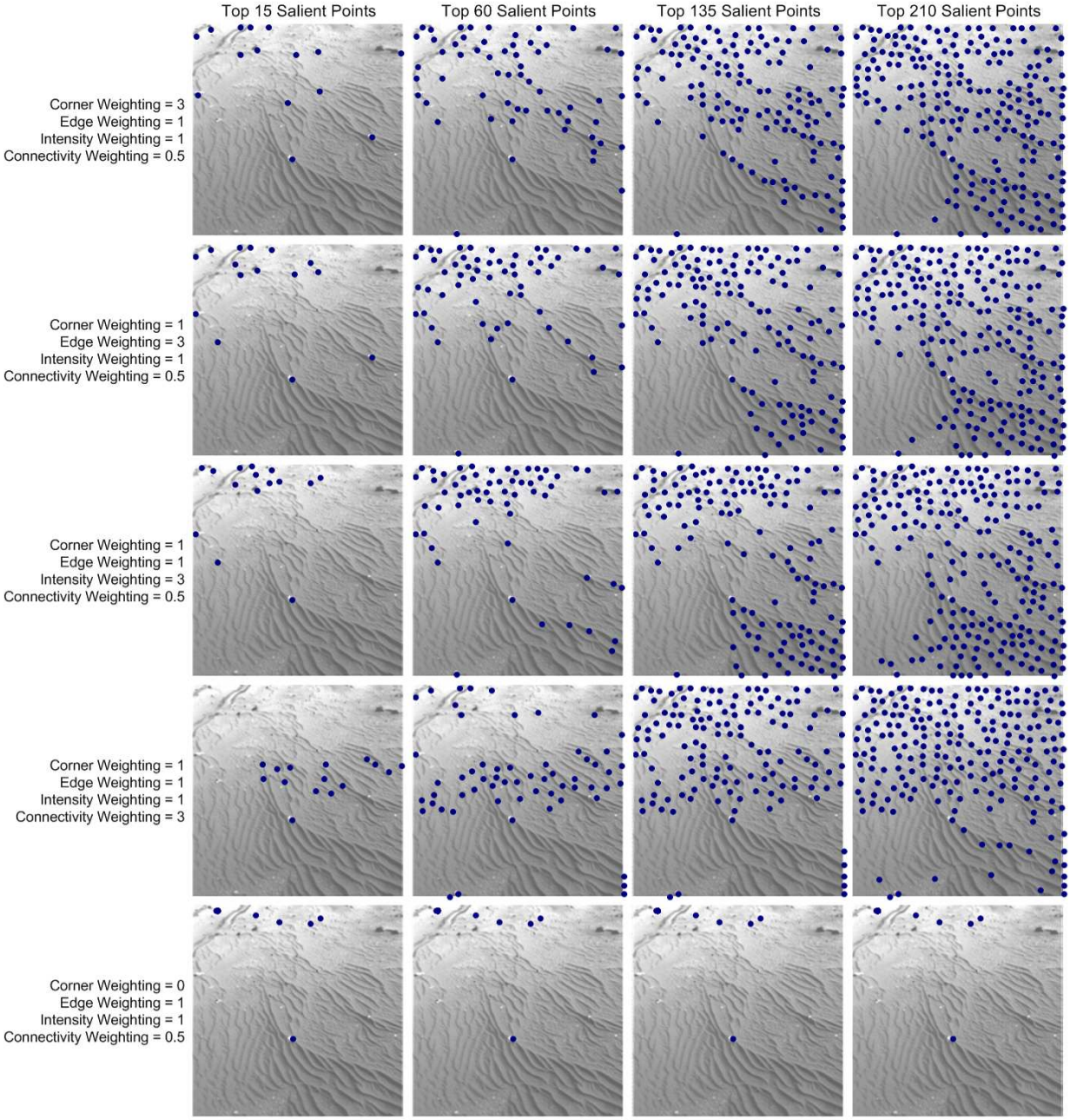


Table 4.2: Duration of scene examination versus feature weighting, autonomous navigation application.

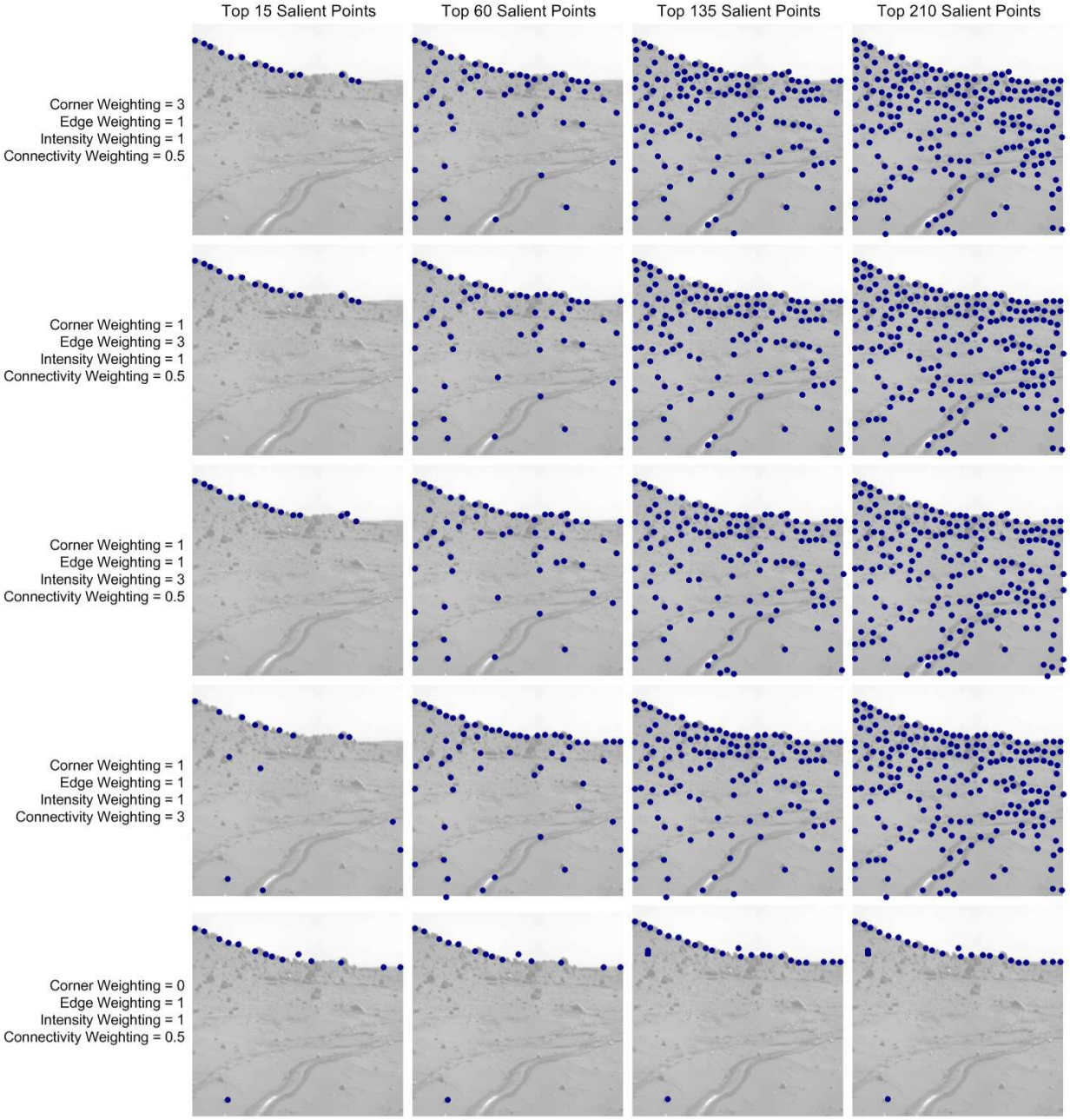


Table 4.3: Duration of scene examination versus feature weighting, facial examination (security/biometrics) application.

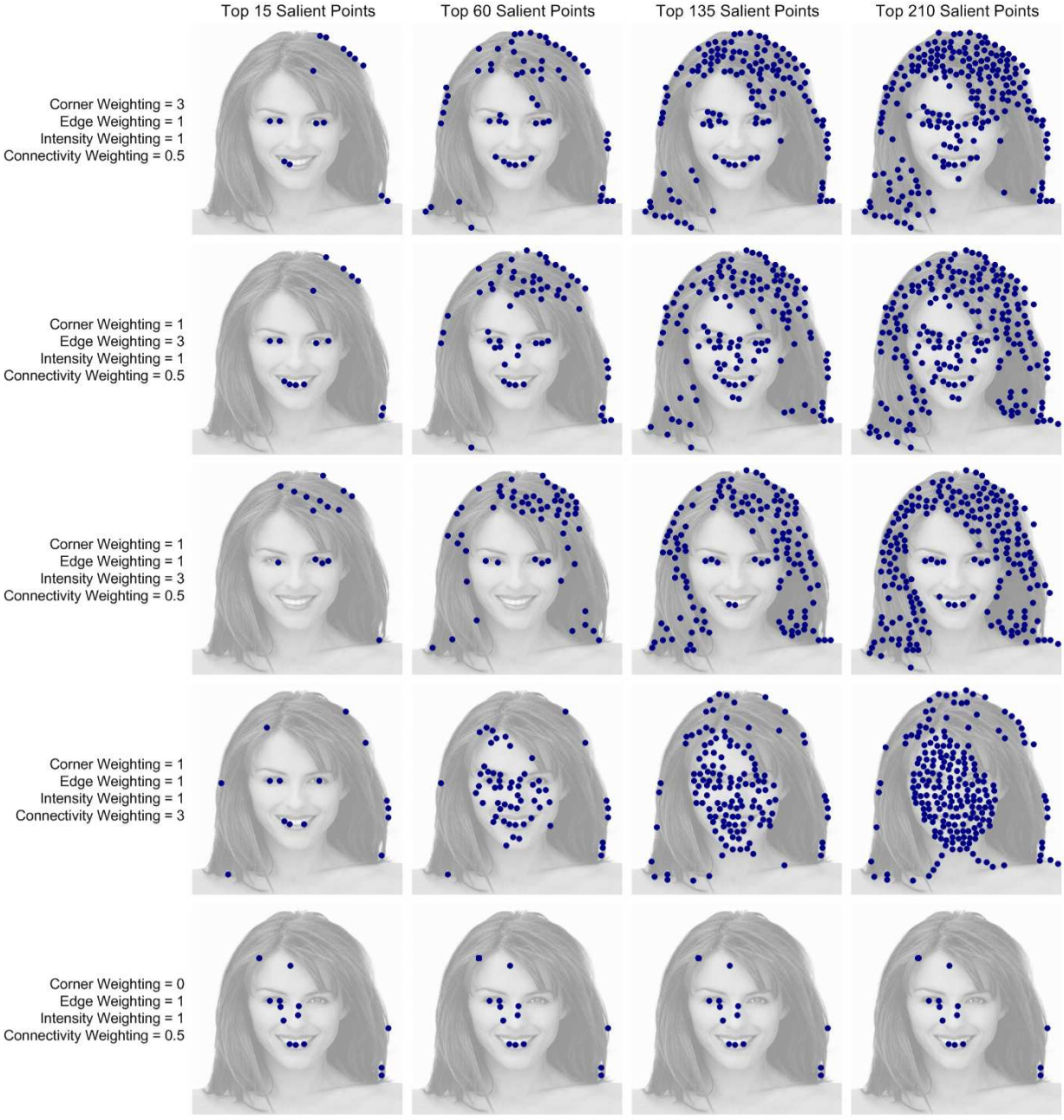
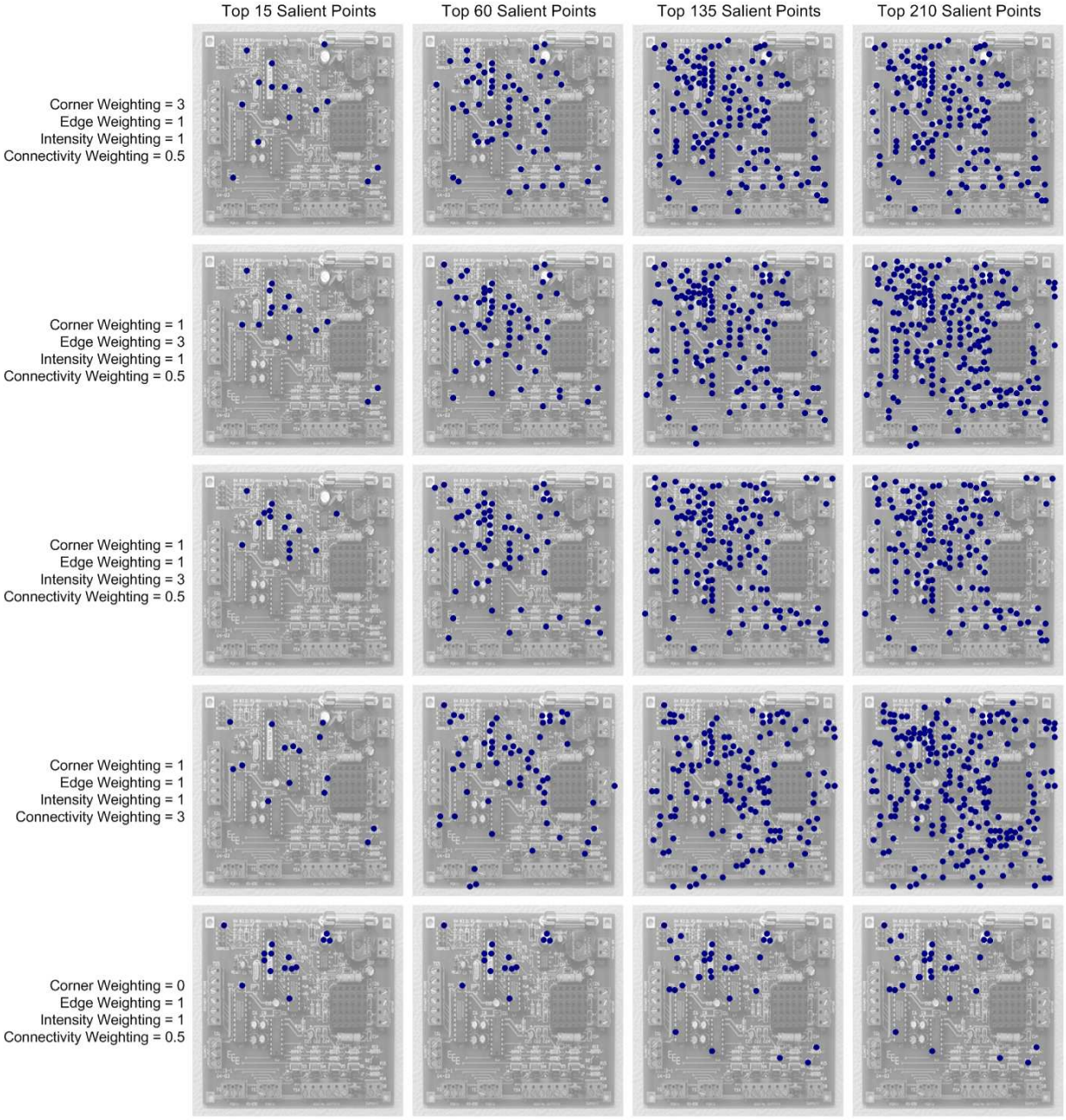


Table 4.4: Duration of scene examination versus feature weighting, industrial inspection application.



Chapter 5

The Empirical Distortion Model

This chapter discusses the implementation of the *adaptation* mode. This mode operates separately from the *scan* and *saccade* modes, and is intended for camera calibration to map image information to physical locations in the real world. Camera calibration is essential for visual servoing, where visual information is used to guide robotic or vehicular pose.

The human vision system (HVS) performs remarkably well under adverse conditions. Even under the most non-ideal of circumstances, we are often able to circumvent obstacles by calibrating our vision system to properly function and interact with the physical world. There are several possible explanations to account for the robustness of the HVS; one such explanation is that the system uses its eye movement capabilities in conjunction with a feedback mechanism, to compensate for low acuity, distorted, or occluded visual data. Jakobson reported on studies of a subject's accuracy in reaching for physical objects, while the subject's eyes are covered by prismatically-displacing goggles. Jakobson concluded that visual feedback allowed the subjects to "subconsciously" calibrate their distorted vision systems for an accurate estimation of real-world locations of physical objects, even in the presence an over 10 degree-shift in displacement [47].

5.1 Scenario of Interest

Consider once again the example of the autonomous Mars Rover. Since it would be difficult to repair physical damage to a vehicle on Mars, there needs to be a 'work-around' such that the

system can still function in the presence of optical flaws. Potential faults that might hinder the performance of the visual navigation system include:

1. Sensor malfunction due to space radiation;
2. Dust particles on the lens; and
3. Thermal degradation of the lens.

Bombardment of ionized particles in space can often alter the performance of image sensors. CMOS sensors tend to be much more resilient against the effects of space particle radiation than CCD sensors. Research to characterize the nature of radiation effects on image sensors can be used to increase the tolerance of sensor designs against charged particles [48],[49],[50].

In the event of the second listed fault, visual obstruction due to dust settling on the lens could be handled by maintaining an internal memory of the dust locations.

The problem of interest for the following sections pertains to the third fault: thermal degradation to the lens. In a harsh environment such as Mars, where the camera is exposed to low atmospheric pressures and solar energy, the lens material could degrade over time. This will cause unknown lens distortions. Although there is much literature on models to characterize common lens distortions [51],[52],[53], this design assumes no knowledge of the nature of the lens distortion in this situation. Therefore, the proposed method examines a way by which the visual navigation system can develop an empirical model of the lens distortion.

5.2 Experimental Setup

The goal of this experiment is to empirically characterize a lens distortion whose nature is unknown, by finding an approximate mapping between the world coordinate system with the image coordinate system, making use of the camera pose with respect to visual targets. The first step is to place a distorted lens on the system. To visually assess the distortion, an image of a Gaussian random target is captured by the camera, shown in Figure 5.1. This image is clearly focused in the central region, and its quality degrades along the edges.

While the image in Figure 5.1 demonstrates the effects of lens distortion, it does not provide useful information for calibration. A less visually impressive, but more useful image, is shown in Figure 5.2a. This is an image of a visual target that contains a pattern of equally-spaced dots. By calculating the centroid of each dot, it was determined that the distance between two dots in the central region differed from the measured distance between two dots captured at the periphery. However, through knowledge of the actual distance between dots, equations can be used to transform the distorted measurement values to corrected measurement values.

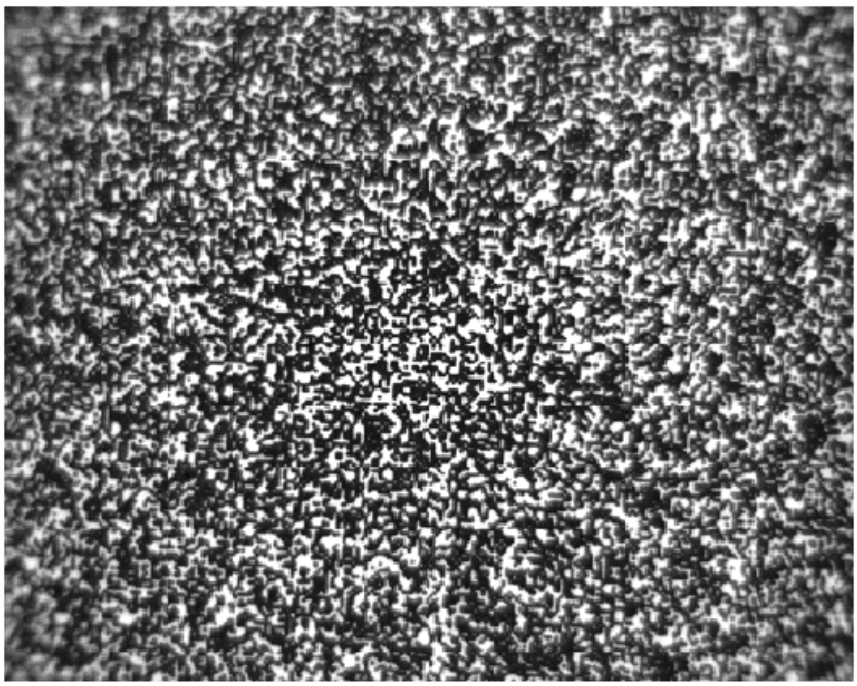


Figure 5.1: Visual target showing effects of lens distortion.

Naturally, the Mars rover can not readily access a target of equally spaced dots. However, the same information could be generated by choosing a point source target (e.g. a clearly defined rock peak, Figure 5.2b and mechanically moving the camera in small, equally-spaced intervals. This idea is the basis of the experiment.

Although Mike's prototype system has a camera mounted on a servomotor, the particular servomotor unit used by the prototype is not suitable for fine movements. Therefore, for the

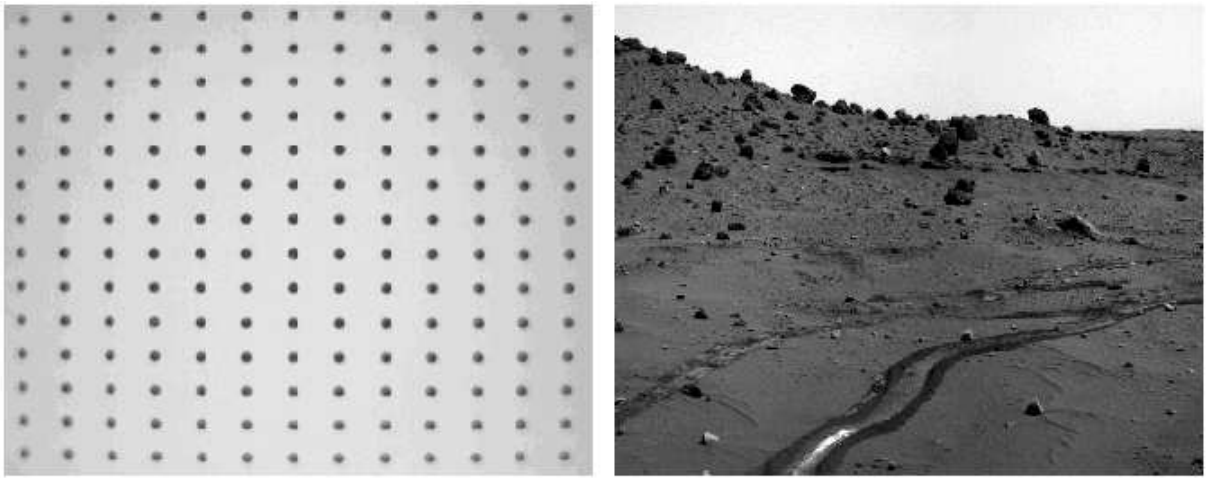


Figure 5.2: Calibration Images: a) visual target of equally spaced dots, b) image of rocks with sharp features on Planet Mars taken by Spirit Rover's Left Panoramic Camera [37].

sake of this experiment, the target is mounted on a translation stage that can be adjusted at μm increments along the vertical and horizontal directions[1]. To artificially recreate the image shown in Figure 5.2a, the target is moved at small increments and the relative motion of the target is measured by determining the centroid location of the point source at each position. A useful model of the distortion effects can be developed with this set of empirical measurements.

5.3 Distortion Model

The lens distortion is clearly non-uniform across the entire visual field (see Figure 5.1). The calibration method assumes that the distortions can be modeled as first order linear effects, as long as the model only describes a small region of the lens at a time. The visual field is arbitrarily divided into a grid of smaller regions, and the first-order distortions in each region are modeled locally. The advantage of this is that the distortion can be modeled linearly with the measured data. The drawback is that there will be discontinuities in the model along the

[1]In addition to providing a higher degree of accuracy and control, use of the precision translator to provide relative motion between the camera and the target keeps the target motion approximately perpendicular to the camera, rather than in an arced trajectory.

regional boundaries. For the sake of simplicity, this is an acceptable caveat of the experiment, as the discontinuities can be reduced through a careful choice of region sizes/shapes, interpolation along borders, and defined overlap between regions.

Table 5.1 presents the notation that will be used in this discussion.

Table 5.1: Distortion Model Notation

Notation	Definition
$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$	The set of distortion coefficients for each region of the lens field.
(x_m, y_m)	The set of measured coordinates of the target's centroid, given in image pixel units.
(x_i, y_i)	The set of ideal coordinates of the target's centroid, as would be measured from an undistorted lens, given in image pixel units.
D_{pixels}	The target's displacement in pixel units.
D_{world}	The target's displacement in world units.
(x_{world}, y_{world})	The target's position in the world coordinate system.
n	Number of measurements per data set.
x_{map}	Scale factor to map horizontal displacements in the real world, to x-axis values in the image coordinate system.
y_{map}	Scale factor to map vertical displacements in the real world, to y-axis values in the image coordinate system.
$\begin{pmatrix} a_m & b_m \\ c_m & d_m \end{pmatrix}$	The empirically obtained set of distortion parameters for a given lens region.
(x_c, y_c)	The corrected values of the target's centroid location, taking into account the regional distortion model. These are in pixel units.
$(x_{world,guess}, y_{world,guess})$	The target's position, estimated by scaling (x_c, y_c) with x_{map} and y_{map}, in world coordinates.

In each region of the visual field, the lens distortion can be described by:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_m \\ y_m \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{5.1}$$

The motion of the target in the real world, relative to the camera, is known to a certain degree of confidence. Treating the world as a 2D field, the world coordinate axes are defined by horizontal and vertical displacements of the target, perpendicular to the camera. The parameters x_{map} and y_{map}, which scale real-world distances to the pixel domain, are first determined for an ideal lens. If the system had a distortion-free lens, measurements of the target's location could be taken at any two positions to find their distance in the x (and y) direction. Dividing the

real-world distance by the pixel distance, this would determine a unit mapping between the real world and the image coordinate system. However, since the actual lens is indeed distorted, it would be better to take a series of measurements, and normalize the real world distances through an average of these measurements. By sampling enough distance measurements from different portions of the lens, the distortion contribution to the final horizontal scale factor, x_{map}, can be minimized. The horizontal distance between two measurements is calculated by:

$$D_{pixels} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{5.2}$$

Since the target's motion is not exactly perpendicular to the camera, it is necessary to take into account the y-components of the motion as the horizontal distance is calculated. Taking an average of the horizontal distance measurements, the real world motion to the image coordinate system is mapped through:

$$\frac{D_{world}}{\sum(D_{pixels})/(n-1)} = x_{map} \tag{5.3}$$

where n is the number of measurements, and x_{map} is the scale factor to map the real world coordinates to ideal pixel coordinates, x_i. (This calculation is repeated for a mapping of vertical displacement in y-values.)

Using a set of displaced measurements for each region (recall that the visual field is divided into regions with separate distortion parameters), the parameters a, b, c, d can be calculated. Based on 5.1:

$$ax_m + by_m = x_i \tag{5.4}$$

$$cx_m + dy_m = y_i \tag{5.5}$$

A minimum of two measurements are required to solve for the unknown parameters. With more measurements, the parameter value solutions can be averaged for a better representation of the region.

Results of an experiment based on this method are presented in Chapter 6.

Chapter 6

Prototype Testing

A test scene consisting of clearly defined white geometric shapes against a dark background was constructed to test the Mike prototype. The testbed is shown in Figure 6.1.

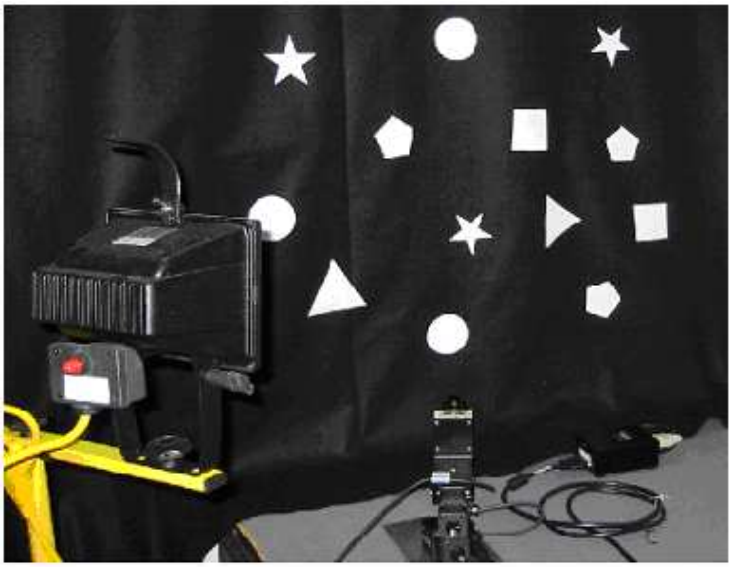


Figure 6.1: Practical experimental setup consisting of clearly defined white geometric shapes against a dark background.

6.1 Experiment #1: General System Function

When Mike faces a new scene, it will scan the scene by capturing a resolution image and generate the saliency map based on the survey image. The camera then serially traverses the saliency list, capturing a fully resolved 32x32 subwindow image (a practical value based on the camera's minimum allowable subwindow size) at each of the salient coordinates. The final output is a reconstruction of the scene, where only the salient areas are detailed in high-resolution. After the saliency list is exhausted for the current scene, the motorized servo shifts the camera's position for a new view. Figure 6.2 shows various frames captured by the system as it saccades through a test scene. The borders around the fovea windows are included in the images for demonstration; in practice, the images would simply contain the low resolution survey image overlaid with the high resolution detail captured by the subwindow. Only corner and edges were considered in this experiment; intensity contrast and connectivity detectors were disabled.

6.2 Experiment #2: Motion Detection and Tracking

Using the test set up from the first experiment, the second experiment tests the scenario when a moving object is introduced to the scene. Initially, all the shapes in the scene are static. As intended, Mike attends to the corners and edges of the shapes, following the priority sequence. Since the corner detector determines the corner coordinates from the 1/4 SXGA low resolution image, there is a level of uncertainty associated with the corner coordinates; furthermore, to account for the object motion, the subwindow size is increased to 80x80 pixels during corner foveation. Figure 6.3 shows a reconstruction of the scene, first shown in low-resolution from the initial scan. The image gains further details about the edges and shapes as the subwindow saccades to conspicuous locations. After a time, the scene changes and the square begins to move. The system detects the object motion and commences tracking of the square. As the square moves across the screen, the subwindow saccades to its corners to track its location and maintain shape information. By foveating the square's corners in high-resolution, the location of the object's centroid can be estimated by averaging corner coordinates.

Figure 6.4 plots an estimation of the object's location based on information returned from

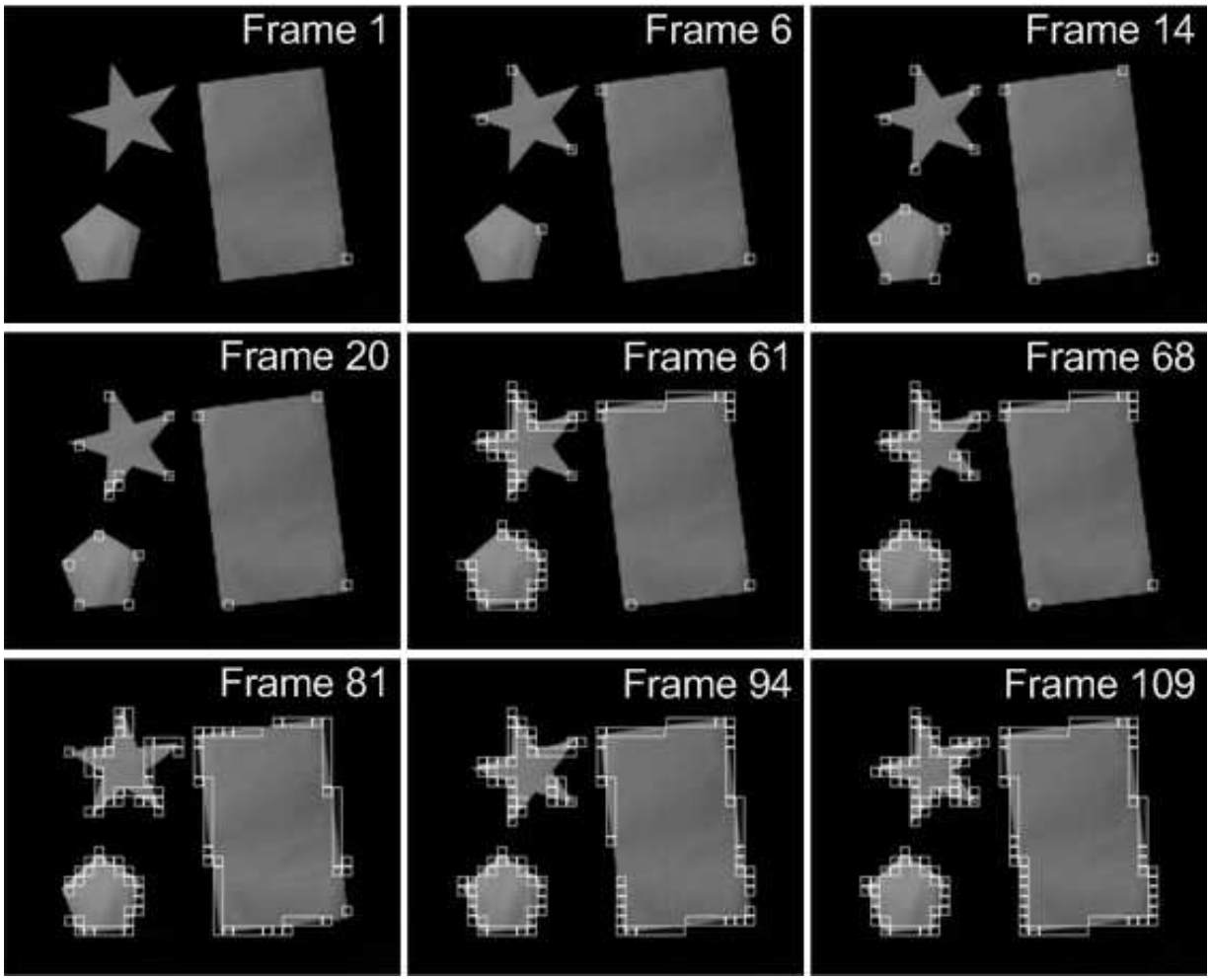


Figure 6.2: Frames captured during saccading. The white borders around the subwindow regions are drawn on the images for demonstration of the process only. In practice, there would be no borders distinguishing the high resolution from low resolution data.

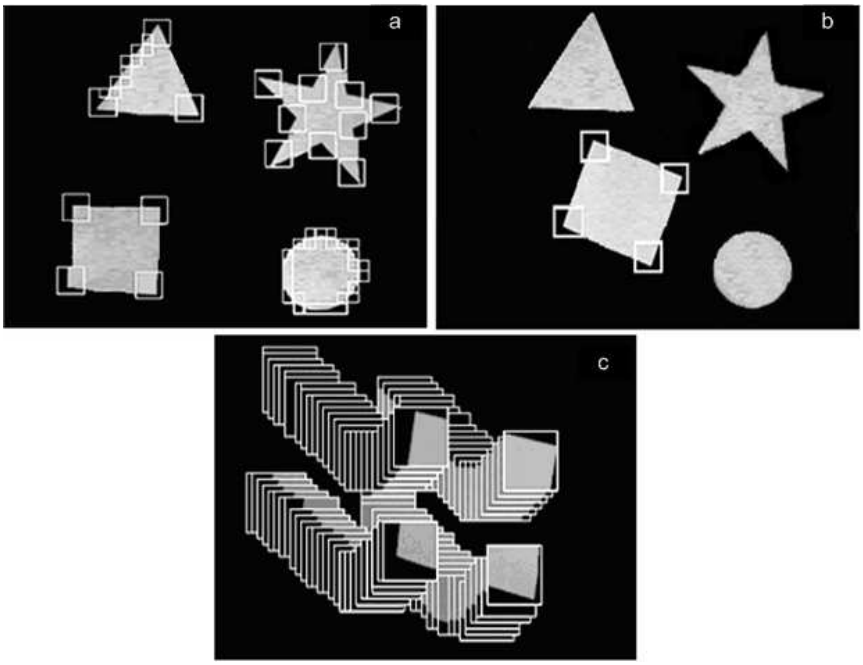


Figure 6.3: a) The high-resolution fovea serially visits the regions of interest (here shown outlined). b) The system detects movement of the square and tracks the object; motion takes priority in saliency considerations and the non-moving shapes are ignored. c) The system foveates the square's corners to track the object's frame-by-frame location.

tracking the square's corners. These approximations are compared against the actual object locations, calculated by a separate setup that took multiple high-resolution images of the entire object at finely distributed locations along the object's path, and applied a centre of geometry calculation to each image.

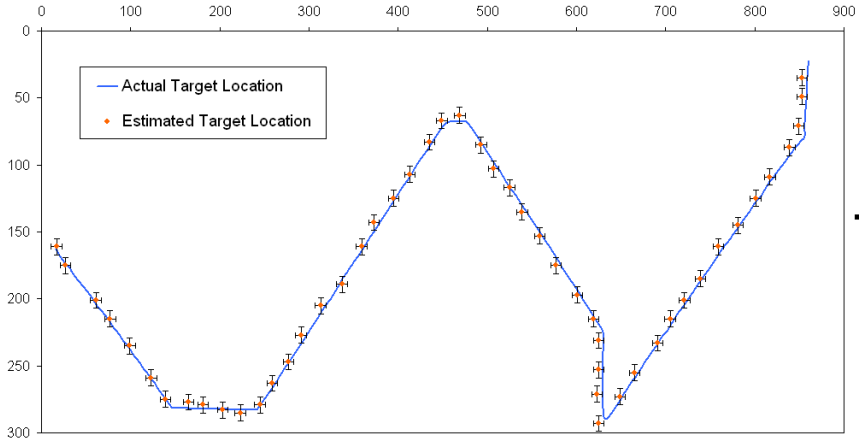


Figure 6.4: The centroid coordinates (in pixel units) of the square as it moves across the field of view in an arbitrary pattern. The line marks the target's actual path, whereas the dots mark the estimated centroid locations based on corner information. Error bars measure 5 pixels from the centroid estimations.

Although there is slight overshoot in the centroid estimations during sudden changes of direction, the estimates are within five pixels of the actual object coordinates, in both x and y directions. It should be noted that in this experiment, the object is moving slowly relative to the camera (1 degree per second). The system's ability to track moving objects is limited by the camera's framerate. Although the camera is capable of achieving an average of 125 fps with the subwindow parameter settings (a theoretical number derived by a calibration utility from the camera's manufacturer [34]), the camera responds slowly to reprogramming of the subwindow's position. These are constraints specific to this particular camera's firmware. However, these limits do not affect the overall potential of an electronic saccadic vision system to achieve rapid selective image acquisition.

6.3 Experiment #3: Adaptation Mode

This experiment tests the creation of an empirical distortion model in the adaptation mode. Using a point source target on a precision translator, 50 target positions were measured at horizontal displacements to determine x_{map}, and 50 additional measurements were taken for vertical displacements to determine y_{map}. For better results, more measurements can be taken at horizontal and vertical paths in different regions of the lens to increase the range of samples. The measured displacements contained some variation, in spite of the uniform target displacements in the real world. Therefore, to improve the accuracy of x_{map} and y_{map}, outlier values beyond 3 standard deviations of the mean were discarded.

The 1280x1024 pixel visual field was arbitrarily divided into 9 equal regions (see Figure 6.5) and the middle region and the four corner regions were parameterized by this experiment. These 5 regions exhibited the least, and the most, amount of distortion, respectively.

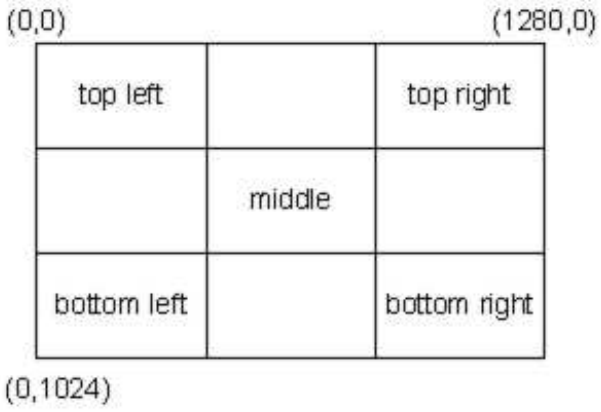


Figure 6.5: Division of the visual field into regions where the distortion effects could be parameterized.

Each set of data (for each region) contained measurements of 100 displacements, providing 101 sets of (x_m,y_m) to calculate the distortion coefficients: a_m, b_m, c_m, and d_m. The set of (x_i,y_i) were the known target displacements in the real world, scaled by x_{map} and y_{map}.

The corrected values of the target's centroid were determined through:

$$\begin{bmatrix} a_m & b_m \\ c_m & d_m \end{bmatrix} \begin{bmatrix} x_m \\ y_m \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \end{bmatrix} \tag{6.1}$$

where the m subscript denotes empirically obtained distortion coefficients.

The set of (x_c, y_c) was then scaled using x_{map} and y_{map} to obtain $(x_{world,guess}, y_{world,guess})$. These final estimated values of the real-world target positions were used to verify the system's ability to determine an object's position in the real world, while equipped with a distorted lens.

Table 6.1 provides a summary of the experimental results. It shows the average target displacements estimated by mapping the uncorrected (unprocessed) centroid positions to the real world coordinate system, and the displacements estimated by the corrected centroid values, calculated using the empirical distortion parameters. Normalizing the uniform target displacements to 1.0 unit per displacement, the error denotes the deviation between the predicted target location and its actual position.

Table 6.1: Calibration Results

Image Region	**Uncorrected Displacement**	**%error**	**Corrected Displacement**	**%error**
Middle	0.99677	0.32%	0.95564	4.44%
Top Left	0.94266	5.73%	1.02668	2.67%
Bottom Left	0.92342	7.66%	1.00213	0.21%
Top Right	0.85494	14.5%	1.07453	7.45%
Bottom Right	0.91296	8.70%	1.07143	7.14%

Interestingly, the distortion model performed better than the raw data for all regions other than the middle, which is least affected by the lens distortions. It can be noted that in the regions most affected by the lens distortions, the modeling of the distortion effects helped to improve the system's ability to predict target positions.

This experiment demonstrates that, if the lens of Mike becomes damaged due to environmental conditions, it is possible for the system to continue to function (that is, provide reliable navigation and target tracking coordinates), by calibrating its interpretation of the measured data through an empirical correction model.

6.4 Simulation vs Practical Implementation

The original intention of the Mike project was to build a saccadic vision system and test the saccadic operations on the prototype, combining both the hardware and software elements of the system. However, several constraints of the hardware components made it difficult to properly utilize the software capabilities of the system. The commercial camera that was used in the prototype was chosen for its programmability and the high framerates that were cited by the camera's datasheet [34]. Although the camera is programmable and has subwindowing capability, it is not intended to be frequently reprogrammed, as the Mike design requires. Thus the camera does not respond quickly to reprogramming of the subwindow's location and size. Furthermore, a bug in the camera's API creates problems in setting the camera's exposure beyond a certain range; the camera cannot be easily programmed to function beyond a small range of exposure times, especially when the subwindow is set to a small size. This required the use of external light sources in the test environment in order to account for the small range of achievable exposures. The testing of the Mike prototype therefore needed to use noisy images due to the uneven light sources and low exposure times.

Due to the problems incurred from using this particular camera unit, much of the testing of the system concept was performed in simulation, rather than in hardware testing.

The current Mike prototype is the result of one potential method of implementation. The Mike system can have several alternate realizations, such as a trainable neural network system, or a system on a chip, where the early stage preprocessing is conducted in hardware. Part II of this thesis discusses design considerations for custom on-chip imaging systems for active vision applications.

Part II

Image Sensor Considerations for Active Vision Applications

Chapter 7

Overview of CMOS Image Sensors

Computer vision is the analysis of electronic visual data, with the intention to extract information that can be used by an artificially intelligent system. Active vision is a branch of computer vision where the available visual information is filtered such that bulk processing is reserved for the most relevant components of the scene.

Prior to the analysis of electronic visual data, a computer vision system must first acquire the data. This is typically achieved using an electronic camera, which consists of a chip containing a photosensitive array. There are two major competing technologies for electronic image acquisition: charge coupled device (CCD) and complementary metal oxide semiconductor (CMOS). Although both types of imaging systems serve the same fundamental purpose, they entail different designs.

CMOS image sensors are a more suitable technology choice for custom on-chip computer vision systems than their charge-coupled counterparts. Although CCDs historically produced better quality images (lower noise and higher uniformity) [33], CMOS sensors boast several advantages:

- lower fabrication costs (CMOS image sensors are fabricated under standard CMOS processes, which are high-volume processes used in the production of computer processors and memories);
- reduced power consumption;

- improved tolerance to radiation[1]; and
- ability to integrate circuits within pixels.

The lower costs and lower power consumption (and hence higher battery life) of CMOS technology motivate market development of CMOS image sensors for consumer products. New fabrication process techniques and photodetector structures (e.g. the pinned photodiode) improve the sensitivity and signal to noise ratios of CMOS sensors, such that their images are competitive with those captured by CCDs [54],[55]. The foremost appeal of CMOS over CCD image sensors for computer vision applications is the integrability of circuits within pixels, thereby allowing random access for selective pixel readout and in-pixel processing.

7.1 CMOS Image Sensor Basics

A solid state image sensor is an array of photosensitive devices that convert optical information into electronic signals. Silicon is commonly used for imaging in the visible spectrum, as it demonstrates high absorption in the visible range, and is low in cost. Photodetectors that can be fabricated in standard CMOS processes include the photodiode, photogate, MOS capacitor, and bipolar phototransistor [56].

7.1.1 Photodiode

A photodiode is a pn-junction diode that is operated in reverse-bias mode; it converts photonic energy into electrical currents. A brief overview of photodiode operation follows.

A photodiode circuits operate under an applied reverse bias voltage to increase the amount of electron-hole generation within the depletion region. The total pn-junction current is a combination of diffusion current (current resulting from electron-hole pairs outside the depletion region), and drift current (current inside the depletion region). Figure 7.1 illustrates the generation of the electric field across the depletion region and movement of charge carriers.

Conceptually, a photodiode can be modeled as a capacitor in parallel with a current source (Figure 7.2). I_{photo} denotes the current resulting from electron-hole generation caused by optical

[1]The radiation hardness of CMOS imagers is of particular utility to space applications.

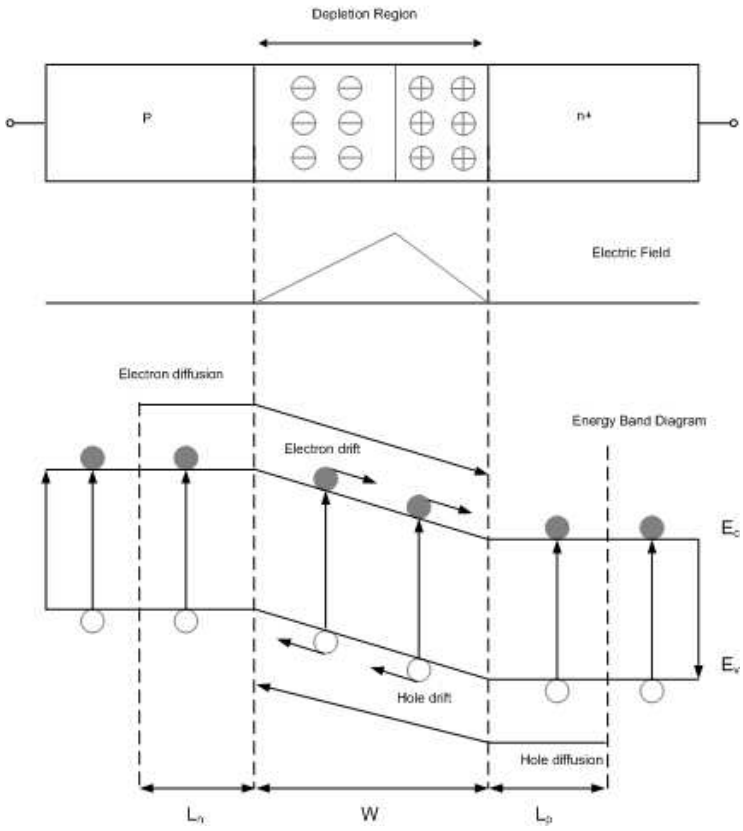


Figure 7.1: Charge carrier motion along a pn-junction [57].

energy. A parasitic diode in parallel with the photocurrent and capacitor models the small thermally-generated current that flows through a real photodiode. This current exists regardless of the presence of illumination, and is thus termed I_{dark}. Since the photodiode normally operates under reverse bias, the dark current can be modeled as a parallel current source, and the net current, I_{total}, is the sum of I_{photo} and I_{dark}.

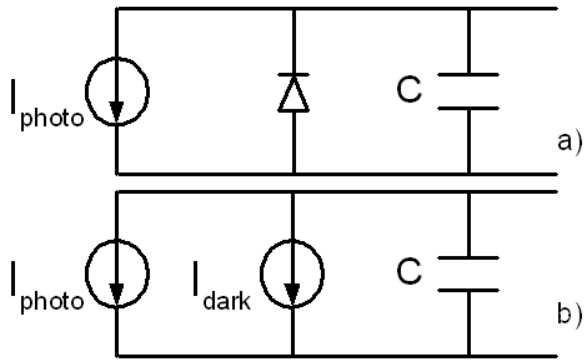


Figure 7.2: Equivalent circuits of a pn-junction photodiode. Part b) models the circuit under reverse bias conditions.

When the reverse-bias voltage is applied to the photodiode, charge collects on its capacitor. This is the *reset mode* of the photodiode operation. After reset, the reverse bias voltage is removed and the photo-generated current in the pn-junction discharges the voltage over a period of time, the integration time, t_{int}. In operation, the photodiode is initially charged to a reset level, V_{RST}. When light illuminates the photodiode surface, the photocurrent will discharge the capacitor (recall $i = C\frac{dv}{dt}$), lowering the voltage from V_{RST} to a reduced level. Since the photocurrent is a function of the intensity and wavelength of the illumination, the stronger the illumination intensity, the faster the capacitor will discharge (see Appendix C for a complete derivation of the relationship between incident illumination and photocurrent). Thus, for a set integration time, the voltage difference between the original applied voltage level and the resulting discharged level provides a measure of the illumination intensity over the photodiode area.

7.1.2 Pixel

Over years of development, pixel circuits have evolved from passive pixel sensors (PPS) to active pixel sensors (APS) [54]. The simplest APS architecture consists of a photodiode and three transistors, and is aptly named the 3T APS (shown in Figure 7.3).

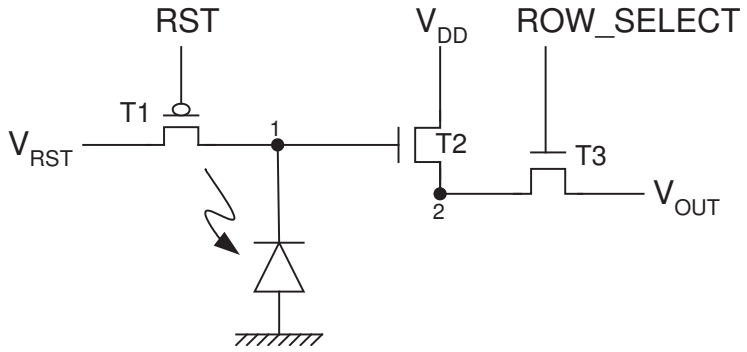


Figure 7.3: 3T Active Pixel Sensor.

When the control signal RST is pulsed low to turn on the transistor T1, the voltage at node 1 (the voltage across the photodiode capacitance) is charged to V_{RST}, which is typically tied to power, V_{DD}. This is the pixel's *reset mode*. When T1 is off (i.e. RST is high), the circuit enters *integration mode*, and the photocurrent discharges the capacitor, lowering the voltage at node 1. T2 acts as a source-follower amplifier that transfers the voltage at node 1 to node 2. It introduces a voltage drop due to its threshold voltage; thus $V_{node2} = V_{node1}$ - $V_{t,source_follower}$. When ROW_SELECT is high, $V_{OUT} \approx V_{node2}$. Figure 7.4 shows a timing diagram of a typical pixel operation scenario. Assuming a relatively small dark current, the slope of V_{node1} during the integration time, t_{int}, is proportional to the photocurrent generated by the photonic flux, F_o:

$$F_o = I_{o,transmitted}/E_{ph} \tag{7.1}$$

$$I_{o,transmitted} = (1-R)I_o \tag{7.2}$$

$$E_{ph} = \frac{hc}{\lambda} \tag{7.3}$$

where I_o is the incident illumination, $I_{o,transmitted}$ is the transmitted incident illumination, R is the reflectivity of the material, and E_{ph} is the energy of a photon proportional to its wavelength. The photocurrent, I_{photo}, that results from F_o discharges the voltage across the photodiode's internal capacitance over the integration time, t_{int}. The difference between the reset voltage, V_{RST}, and V_{OUT} is therefore a measure of the intensity of the incident light on the pixel surface. Appendix C provides a set of characteristic equations that describe the photodiode operation in more detail.

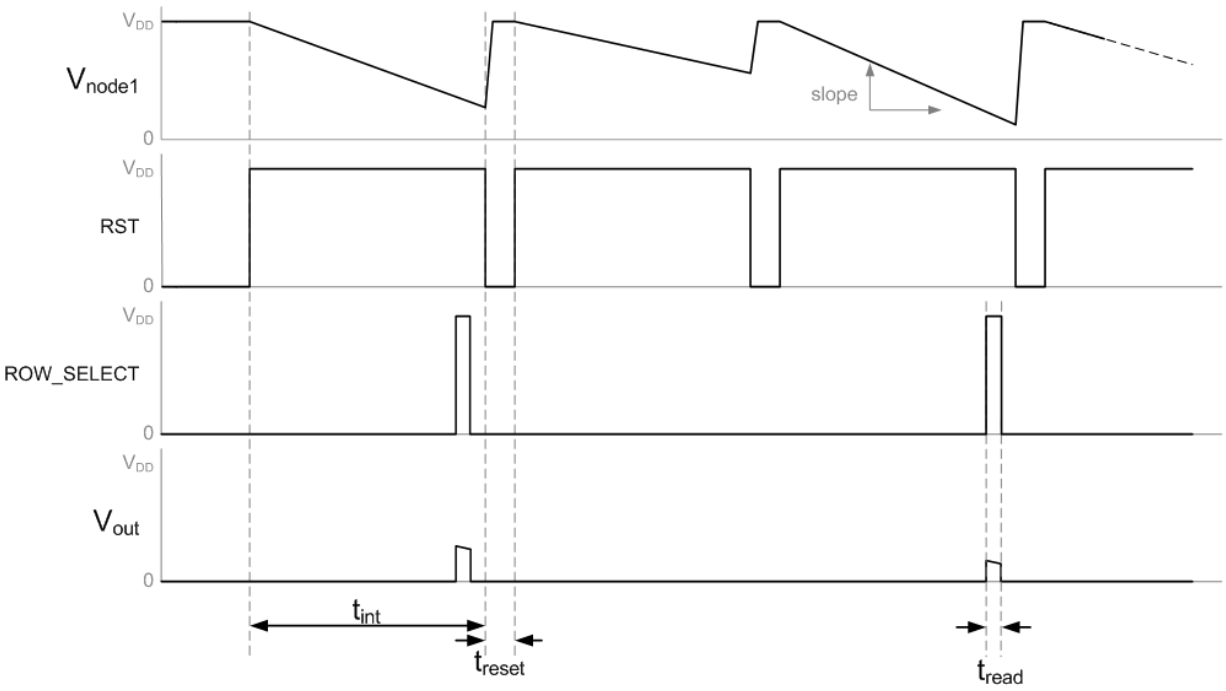


Figure 7.4: Timing diagram for a 3T APS.

7.1.3 Fill Factor

The pixel's fill factor is the percentage of the pixel occupied by the photodiode; it indicates how much of the pixel geometry is responsible for the detection of light. Ideally, the fill factor should be as large as possible in order to reduce the integration time (the larger the photosensitive area, the faster the photodiode response). On the other hand, the integrability of transistors within the

pixel offers several useful functions, such as the random accessibility of pixels and the potential for in-pixel processing. The tradeoff from the presence of transistors in the pixel is reduced fill factor.

7.1.4 Dynamic Range

Dynamic range describes the range between the minimum detectable illumination level and the maximum detectable illumination level. A pixel is said to 'saturate' when the incident light intensity causes the voltage across the photodiode to drop to V_{ss} during the integration time. The dynamic range of a typical 3T APS structure is determined by the voltage swing between V_{RST} (which is typically V_{DD}) and ground, V_{SS}. As technology scales (i.e. transistors become smaller), the power supply also reduces (e.g. V_{DD} = 3.3V for 0.35μm technology and V_{DD}=1.8V for 0.18μm techology). This in turn compresses the voltage swing across the photodiode, imposing a limit on the number of decipherable intensity levels. Futhermore, the voltage drop across the source follower diminishes the range of voltages that can be detected at the pixel's output. Unfortunately, although power supply reduces with the scaled technology, the threshold voltage of the source follower does not reduce as aggressively. Thus, dynamic range is limited at the low end by dark current, and at the high end by the source follower.

While technology scaling is advantageous for fill factor, (because as transistors become smaller, they occupy less percentage of the pixel area), technology scaling limits dynamic range.

7.1.5 Noise

Noise in an image sensor array causes effects such as fixed pattern noise (FPN) and photo-response non-uniformity (PRNU). Common noise sources include: dark current (thermal noise), reset noise, flicker noise, and read noise.

Dark current , which contributes to shot noise (random noise), results from the generation of electron-hole pairs from thermal energy. It exists independently of illumination and is due to the inherent statistical behaviour of an electron.

Reset noise is often called kTC noise because it is a function of the sensor's temperature and capacitance:

$$\langle n_e \rangle = \frac{\sqrt{kTC}}{q} \tag{7.4}$$

Reset noise causes variance in the voltage to which a photodiode is initially charged.

Flicker noise is inversely proportional to frequency and dominates at low frequencies. Its root cause is not well understood [56], but it causes conductively fluctuations along junction between metals and semiconductors.

Read noise is the culmination of noise along the read path, which might include a chain of amplifiers and storage circuits.

7.1.6 Sensitivity/Response

The bulk of photodetection in a photodiode is achieved in the depletion region along the pn junction. Although a percentage of all visible wavelengths can penetrate the photodiode surface, blue light tends to stop near the surface (due it its short wavelength) and red light is absorbed in regions generally deeper than the depletion region (due it its long wavelength). There are various design and process techniques that can be used to improve the photodiode's response to red and blue light. The depth of the depletion region can be increased by fabrication on a wafer with a thicker epitaxial layer [58]. The pinned photodiode structure, which employs modifications to the CMOS implanting process, can also be used to improve quantum efficiency (especially for blue light) [59].

7.1.7 Sensor Array

In a typical CMOS pixel array readout scheme, all the pixels along a column share an output bus, and the ROW_SELECT signal controls which row drives the bus at a given time. A readout circuit at the end of the column bus might consist of various combinations of circuit blocks, such as an analog sample and hold circuit, an analog-to-digital converter (ADC), and perhaps digital memory. In the readout scheme for a typical mxn array, one row is enabled at a time, and there is a reset and integration cycle for each row readout. Therefore, for the mxn pixel array, one full frame requires m reset and integration cycles (see Figure 7.5).

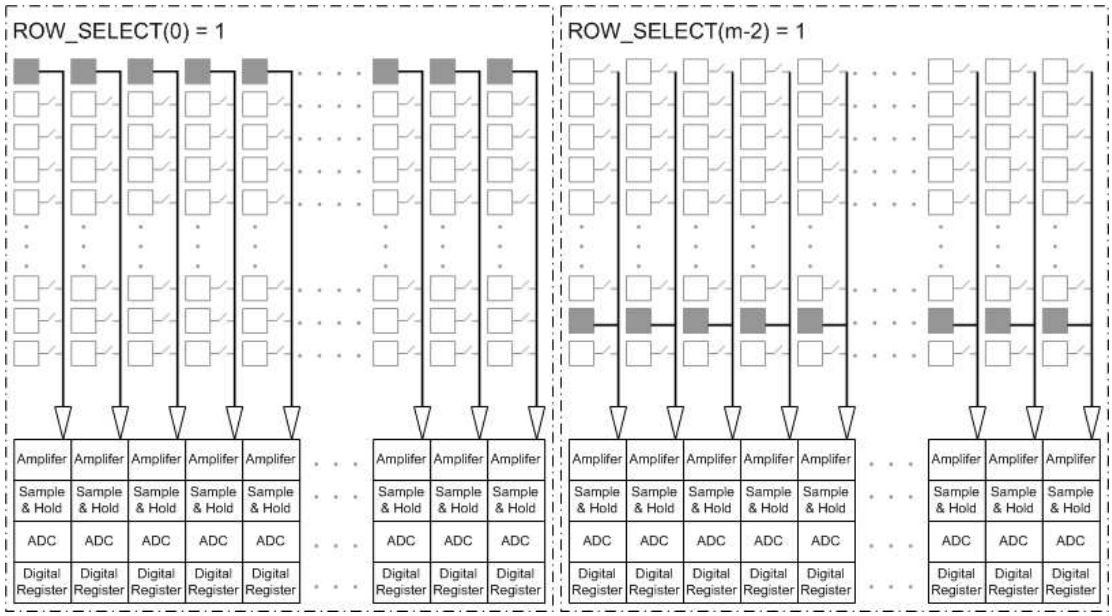


Figure 7.5: Readout scheme for a 3T APS. The column read lines are shared by all the rows. Although several (or all) columns can be read in parallel, each row is read one at a time.

7.1.8 Region of Interest

A region of interest (ROI) is a subwindow that containing information of interest. For example, in a laser tracker, the ROI would contain the laser spot, or in the Mike system, the ROI would contain a salient area. Rather than waste time and resources on a full frame readout, a selected

subgroup of pixels can be isolated by addressing individual row and column select lines (see Figure 7.6).

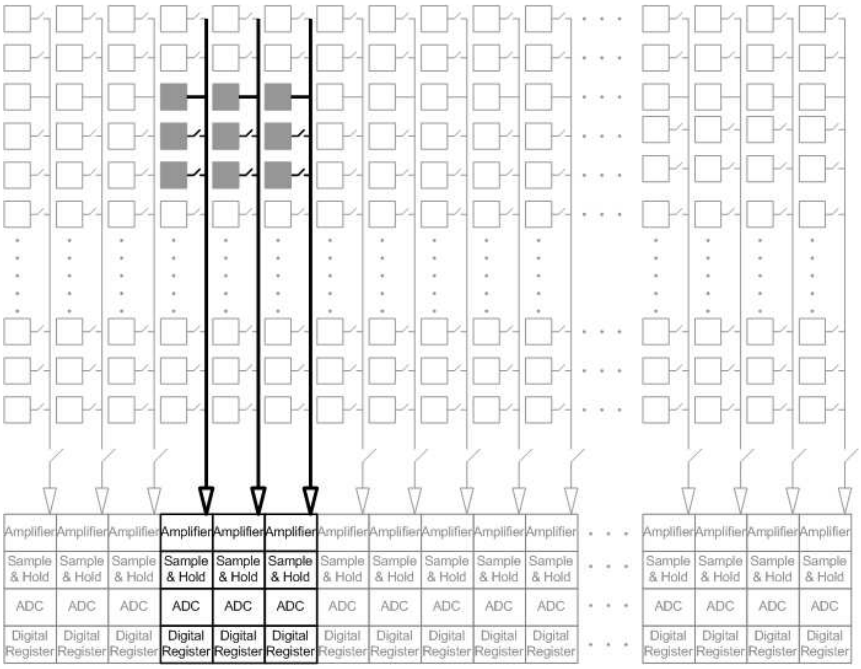


Figure 7.6: The subwindow readout takes three read cycles. During all three cycles, COLUMN_SELECT(3..5) = 1, and on the first read cycle ROW_SELECT(2) = 1, on the second read cycle ROW_SELECT(3) = 1, and on the third read cycle ROW_SELECT(4) = 1.

ROI is typically considered to have more intelligence than simple subwindowing; it includes the choice of the subwindow parameters (i.e. size and location). A 'smart' CMOS imager therefore includes a means by which the ROI is determined.

Chapter 8

Custom CMOS Imagers for Active Vision

This chapter will discuss two functional requirements for custom CMOS image sensors for active vision applications: region of interest and high dynamic range.

8.1 Region of Interest (ROI)

For the purpose of fast readout, many machine vision applications (such as 3D imaging, etc.) require custom image sensors with region of interest (ROI) capability. As the name implies, a region of interest is a subwindow, whose size and location are chosen because it contains information of value. The design problem lies in how to chose the ROI's size and location. For example, the commercial PixeLink camera allows selective subwindow readout, but the onus is on the user to specify the parameters of the subwindow (i.e. size and location). Without an a priori knowledge of the contents of the scene, it is difficult to determine those parameters.

There are various potential solutions to the ROI problem, including Burn's cumulative cross section method to estimate the region of interest around a laser spot [60], De Nisi's region of interest identification of a laser spot using a two-row array [61], and Schrey's skip logic technique [62]. In Burn's cumulative cross section scheme, he projected the laser spot shape onto single row

and column vectors, creating a rectangular description of the spot [60]. In De Nisi's work, two rows are stacked; the top row consists of larger pixels with faster response than the bottom row. The general region containing the spot is detected by a winner-take-all circuit attached to the top row. The corresponding region is then read out of the higher-resolution but slower responding bottom row [61].

Schrey's skip logic is an on-chip method of addressing 32x32 pixel regions of interest, whose locations are determined externally [62]. The skip logic reduces the burden of decoding every pixel's address in the sensor by grouping pixels into minimum sized subwindows. The design's name comes from the skipping pattern of addressing the subwindows. Logic corresponding to each pre-defined grouping of subwindows, called 'skip blocks', determine whether or not a skip block is valid for readout. High level logic scans the skip blocks to search for valid skip blocks for reading. Multiple skip blocks can be combined for readout to scale the size of the region of interest. According to Schrey, skip blocks can be quickly addressed and their address decoders would occupy less chip space than a decoder for fully random addressable subwindows. Since Mike uses a pre-defined grid of sectors to determine the foveal locations (see Section 3.9) this ROI addressing technique would be particularly suitable for a custom chip for Mike. The saliency map would provide the ROI locations and the skip logic could be used to quickly address those regions.

8.2 Dynamic Range

A large range of detectable intensities is necessary for computer vision applications. As technology scales, so does power supply voltage. This has unfortunate consequences for dynamic range because a reduced power supply reduces the pixel's voltage swing. This section discusses a pixel design technique that addresses the issue of dynamic range.

8.2.1 Pulse Frequency Modulation Pixels

The 3T APS readout scheme described in Chapter 7 is typical: upon reset, the photodiode capacitance is charged to the reset voltage level, and upon integration, the photocurrent discharges

the capacitance at a linear rate proportional to the incident light intensity. However, the ability to detect a wide range of light levels is hindered by the reduction in supply voltage as technology scales. This is true for a typical APS readout.

In a pulse frequency modulation (PFM) scheme, the source follower amplifier (T2 of Figure 7.3) is replaced by an analog comparator. This comparator evaluates the photodiode voltage level against a pre-defined threshold voltage. When the photodiode voltage drops below the threshold, the comparator output triggers the reset transistor (T1), thus recharging the photodiode. Figure 8.1 shows two possible implementations of a PFM pixel. Therefore, rather than a single linear discharge of the photodiode during the traditional integration time (t_{int}), the photodiode discharges and recharges multiple times. However, as the discharge slope is still proportional to the incident light, the reset frequency is also proportional to the light. Therefore, the average intensity of light during the 'integration time' is encoded in the pulse frequency. Figure 8.1a shows the pixel proposed by Wang [57], which contains an n-bit counter within the pixel; the output is a n-bit digital bus. Figure 8.1b shows the same pixel with a single analog output, which is intended to connect to a column-wise counter when the row is selected; in this implementation, the n-bit counter is shared by the entire column. The first implementation has the advantage of in-pixel analog to digital conversion (ADC), but with a severe loss of fill factor (due to the area occupied by the counter). The buffer along the feedback path restores voltage swing of the comparator output. The second implementation performs column-wise ADC and therefore takes up less area within the pixel. Since the analog output is read and converted one row at a time, the gate signal for T1 is function of ROW_SELECT and the comparator output, thereby inhibiting the constant switching and power drain of the pixel during the read cycles of other rows.

The output pulse frequency is dependent on the capacitance across the photodiode, the photocurrent, the dark current, and the reference voltage:

$$f = \frac{i_{photo} + i_{dark}}{C(V_{DD} - V_{ref})} \tag{8.1}$$

Figure 8.2 shows the timing diagram for a PFM pixel based on the implementations described in Figure 8.1.

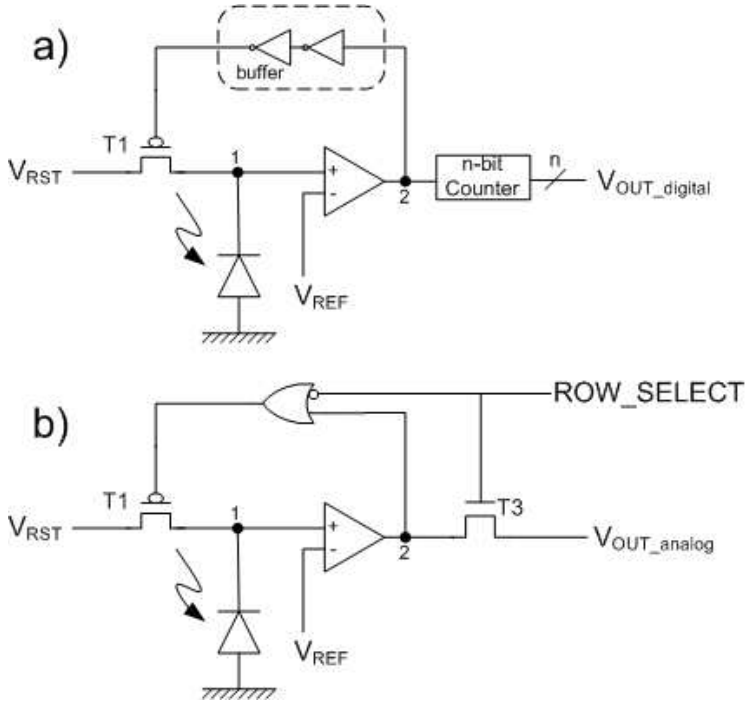


Figure 8.1: Two implementations of a PFM pixel. a) Wang's pixel [57]. b) Modification to Wang's pixel, where the counter is shared along the column, and the integration operation is suspended when the row readout is inactive.

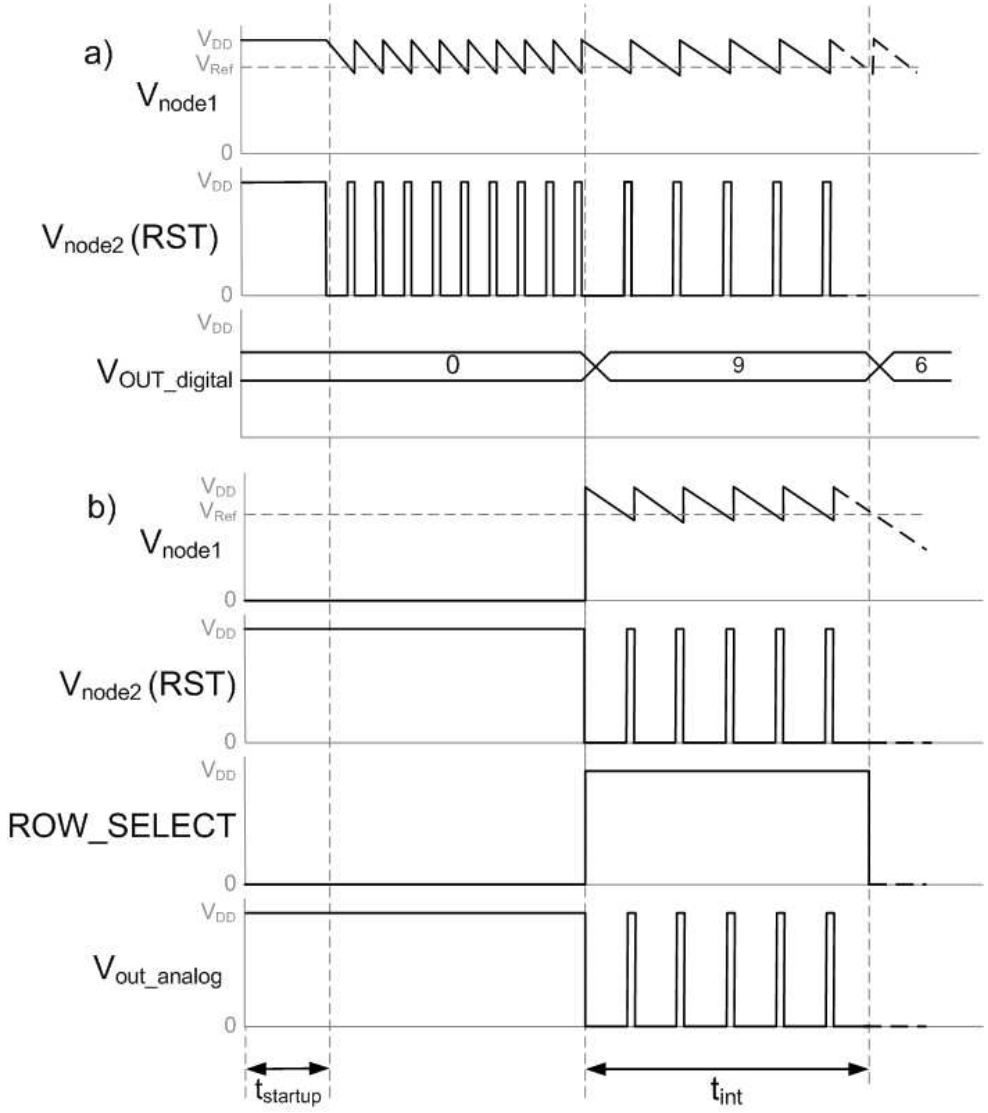


Figure 8.2: Timing diagram for PFM pixels. a) Timing diagram for pixel shown in Figure 8.1a. b) Timing diagram for pixel shown in Figure 8.1b; this design assumes that the digital counter is shared by the column.

8.2.2 Comparator Designs

The main design consideration in the choice of comparator architecture is area. In her evaluation of comparators for the pixel shown in Figure 8.1a, Wang considered the use of four comparator architectures: i) a two-stage differential comparator biased in the subthreshold region (Figure 8.3a), ii) a symmetrical operational transconductance amplifier (OTA) (Figure 8.3b), iii) a symmetrical OTA with a feedback network (Figure 8.3b), and iv) a hysteresis-controllable Schmitt-Trigger (Figure 8.3c) [57]. Each circuit was intended for operation at 1.2V power supply (to reduce power consumption), and smaller transistor dimensions (for smaller area) were selected at the expense of gain, speed and matching (between different pixels). This thesis presents a fifth comparator design, using two simple back-to-back inverters, Figure 8.3d.

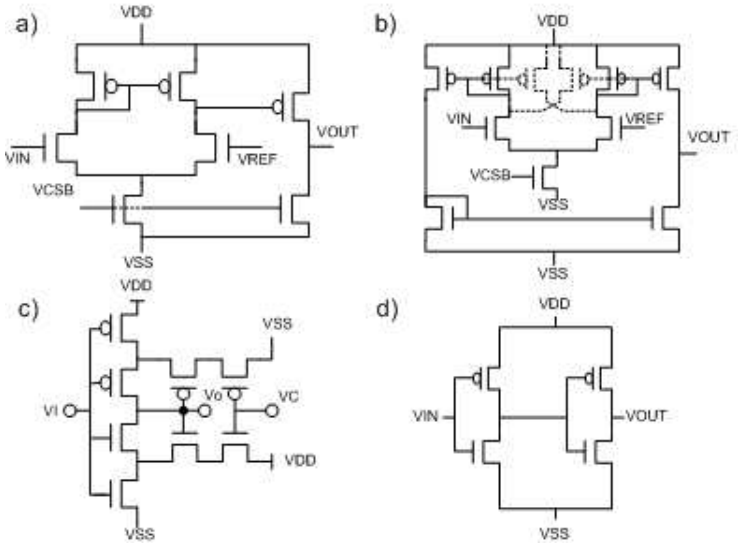


Figure 8.3: Various comparator designs for use in a PFM pixel. a) Two-stage differential comparator biased in the subthreshold region. b) Symmetrical operational transconductance amplifier (OTA), can be implemented with feedback transistors (dotted lines). c)Hysteresis-controllable Schmitt-Trigger. d)Inverter.

8.2.3 Using an Inverter as a Comparator

A static CMOS inverter is the simplest CMOS circuit; it is formed by a pmos and an nmos transistor joined at their gates (input), and connected between the pmos drain and nmos source (output). Its switching threshold voltage is the brief point at which both the pullup (pmos) and pulldown networks (nmos) conduct current (i.e. are both in the saturation region) [63]. Graphically, it is the intersection of the voltage transfer characteristic curve with the line $V_{OUT} = V_{in}$ (see Figure 8.4).

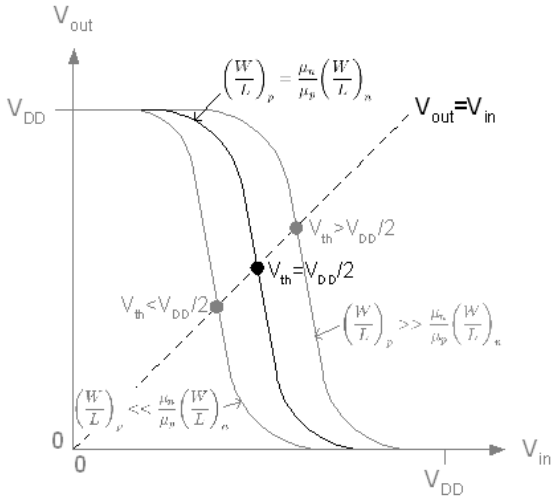


Figure 8.4: Voltage transfer characteristics of a CMOS inverter.

An input signal below the threshold voltage will result in a high output, while an input above the threshold voltage will result in a low output. The switching threshold is given by [64]:

$$V_{th} = \frac{V_{DD} - |V_{tp}| + \sqrt{k_n/k_p}V_{tn}}{1 + \sqrt{k_n/k_p}} \tag{8.2}$$

where $V_{tn,p}$ are the transistor threshold voltages of nmos and pmos transistors, respectively, and:

$$k_{n,p} = \mu_{n,p} C_{ox} \left(\frac{W}{L}\right)_{n,p} \tag{8.3}$$

In a matched inverter, the relationship between the pmos and nmos transistor size is [64]:

$$\left(\frac{W}{L}\right)_p = \frac{\mu_n}{\mu_p}\left(\frac{W}{L}\right)_n \tag{8.4}$$

When the pullup and pulldown networks are matched, the switching threshold becomes $\frac{V_{DD}}{2}$. Typically, this is ideal. However, it is possible to set the switching threshold to higher (or lower) than $\frac{V_{DD}}{2}$ by adjusting the ratio of transistor widths between the pmos and nmos. An inverter can therefore be used as a voltage comparator, using its switching threshold as the reference voltage V_{ref}. The static inverter has several attractive circuit qualities: fast switching, full voltage swing at the output, low power, and occupation of small area. However, inverters (in their traditional forms) are typically unsuitable as analog comparators because the reference voltage is determined by the physical dimensions of the circuit devices and therefore cannot be easily changed. Also, it would take overly long for the photodiode to discharge from V_{DD}-V_{tp} to below $\frac{V_{DD}}{2}$, which is necessary in order to trigger the reset in the PFM scheme. The following two subsections will discuss design strategies that will allow an inverter to be used as a comparator in the PFM pixel. The first strategy is an inverter design with programmable switching threshold. The second technique varies the initial pixel reset voltage in order to reduce the voltage difference between V_{RST} and V_{th}.

Inverter with Programmable Switching Threshold

If the transistors in the pullup and pulldown networks share the same transistor length (which is typically the case), then Equation 8.2 becomes [65]:

$$V_{th} = \frac{V_{tn} + \sqrt{\frac{\mu_p W_p}{\mu_n W_n}}(V_{DD} - |V_{tp}|)}{1 + \sqrt{\frac{\mu_p W_p}{\mu_n W_n}}} \tag{8.5}$$

It is important to note that W_p and W_n denotes the effective width of the pullup and pulldown networks. Therefore, if the (W_p/W_n) could be adjusted, then the switching threshold could be made programmable.

Segura proposed a variable threshold voltage inverter that consisted of a single pmos transistor

in the pullup network, and multiple nmos transistors in the pulldown network [65]. The parallel-connected nmos transistors could be individually turned on or off, thereby adjusting the size of the pulldown network (the effective width is the sum of the individual widths). Therefore, the design had a fixed W_p and a variable $W_{n,effective}$. The threshold voltage would be a function of the number of nmos transistors that are enabled in the pulldown network. If each of the nmos transistor were a different width, then there would be 2^{n-1} values of V_{th} [65]:

$$V_{th,i} = \frac{V_{tn} + \sqrt{\frac{\mu_p W_p}{\mu_n (W_n + \sum_{j=1}^{n-1} p_j W_{n,j})}}(V_{DD} - |V_{tp}|)}{1 + \sqrt{\frac{\mu_p W_p}{\mu_n (W_n + \sum_{j=1}^{n-1} p_j W_{n,j})}}} \tag{8.6}$$

Segura's design single-gate nmos and pmos enhancement transistors, and double-gate nmos transistors in the programmable portion of the pulldown network [65].

The proposed inverter circuit uses the same concept as Segura's design, implemented in a single-poly process. Although the effective width can be varied for either (or both) the pullup or pulldown network, the transistors in the pullup network are typically much larger than those in the pulldown network (due to the difference in carrier mobilities). Therefore, the pulldown network is a better choice for the implementation of the variable width. Each programmable portion of the circuit requires two transistors: a transistor to provide the pulldown path to ground, and another transistor to enable/disable the pulldown path to ground. This arrangement has two configurations, shown in Figure 8.5: i) the enabling transistor is connected to the pulldown transistor's gate, and ii) the enabling transistor's drain is connected to the pulldown transistor's source.

Inverter with Variable Reset Voltage

Alternatively, a regular inverter can be used, and the range of its input voltage (V_{node2} in the PFM pixel) could be modified. Typically, a pixel's reset transistor is tied to V_{DD}, to maximize the voltage swing between V_{SS} and V_{DD}. However, the PFM scheme intentionally only utilizes a small fraction of the voltage swing, thus a maximum swing is unnecessary. Assuming that the photodiode discharges at an approximately linear rate, the photodiode in a PFM pixel can

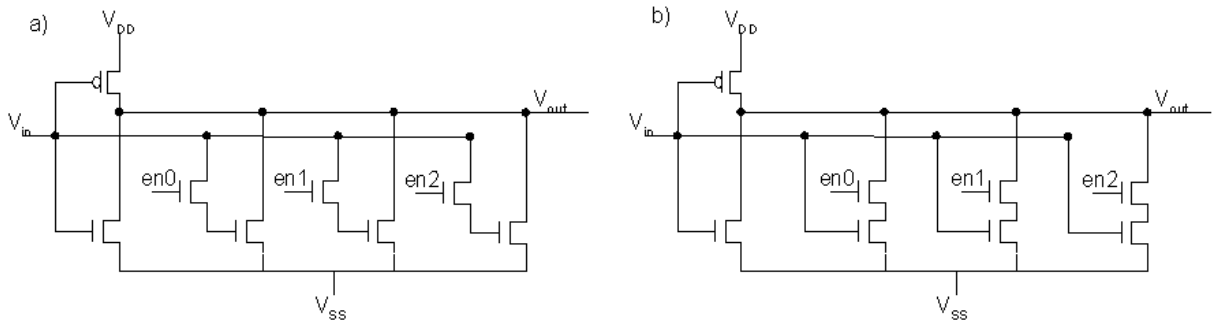


Figure 8.5: Two configurations of an inverter with programmable switching threshold.

theoretically be reset to any value above V_{SS}, and the reference threshold (for the comparator) can be assigned any value between V_{SS} and V_{RST}. Therefore, V_{RST} can be assigned a value much lower than V_{DD}, and an nmos transistor can replace the traditionally pmos reset transistor.

Although this strategy is conceptually very simple, its implications towards the PFM implementation is extremely useful, as it allows an inverter to be used as the comparator. Similarly, this technique could be used for a PFM pixel using the Schmitt trigger. Wang reported that the Schmitt trigger displayed better linearity than the analog comparators, but did not achieve the same dynamic range because it required excess time for the photodiode to discharge to the Schmitt trigger's threshold level [57]. Using Wang's pixel simulation testbench, a fifth implementation of her PFM pixel (Figure 8.1a) is implemented using an inverter comparator with reduced reset voltage. The following sections present the results of simulations on Wang's four comparators and the proposed inverter comparator with variable reset voltage.

Simulation Results

Each comparator (from Figure 8.3) was tested using the same Cadence testbench, which models the pixel shown in 8.1a[1]. The photodiode was modeled as a current source in parallel with a 50fF capacitor and V_{OUT} was connected to a 3-bit static counter. Table 8.1 lists the average pulse

[1]Actually, when the inverter is used as the comparator, it is unnecessary to include the feedback buffer, as the inverter output will have full voltage swing. However, in order to compare results from the different implementations, all the designs are tested with the same pixel architecture. In an actual implementation, the results from the inverter comparator would be better than the results reported here because the buffer would be removed from the feedback path.

widths and reset delays resulting from various levels of photocurrent. The pixel's frequency is the inverse of the pulse width.

Table 8.1: PFM Simulation Results

	Symmetrical OTA OTA		Symmetrical OTA with Feedback		Two Stage Comparator		Schmitt Trigger		Inverter	
	Pulse Width [μs]	Reset Delay [ns]	Pulse Width [μs]	Reset Delay [ns]	Pulse Width [μs]	Reset Delay [ns]	Pulse Width [μs]	Reset Delay [ns]	Pulse Width [μs]	Reset Delay [ns]
10^0	2487.5622	5.100	2109.7046	5.651	2481.3896	11.265	20000.0000	2.006	1439.0000	1.713
10^1	288.4000	5.628	262.9000	5.870	269.9784	10.803	1663.8935	2.006	93.2500	1.700
10^2	8.7900	5.127	27.2700	7.003	27.4997	13.889	162.9992	1.852	8.4890	1.735
10^3	3.0370	5.969	2.9390	11.640	2.9000	27.624	16.3591	1.852	0.9327	1.671
10^4	0.4036	9.189	0.4994	24.850	0.3500	59.414	1.7031	2.006	0.0941	1.624
10^5	0.0763	17.380	0.1113	45.110	0.0700	86.574	0.2003	1.698	0.0124	1.597

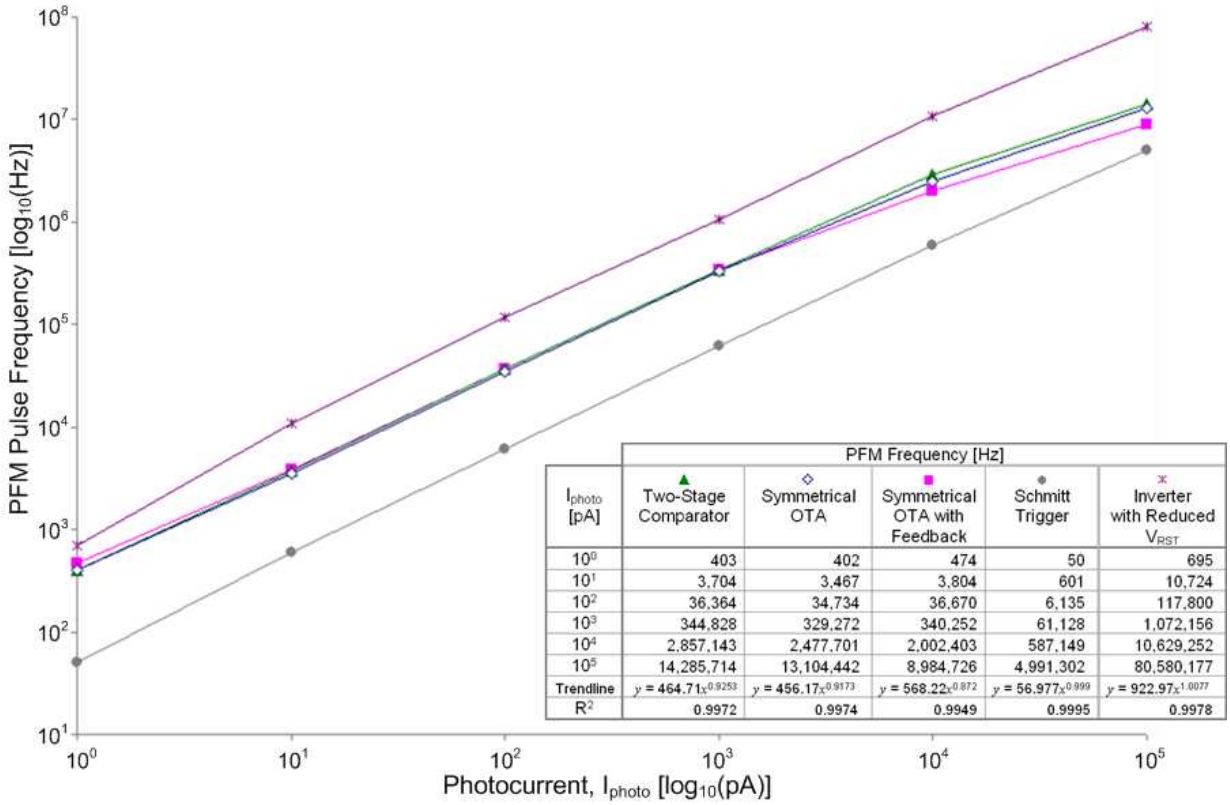
Performance (Dynamic Range)

The dynamic range is the range of intensities that can be detected by the pixel. In a typical APS pixel, the dynamic range is the number of light levels that can be encoded between V_{SS} and V_{RST} during the integration time. When the incident illumination is high, the photocurrent is large, causing the photodiode to discharge quickly. Therefore, it is necessary to reduce the integration time in order to prevent the pixel from saturating (discharging completely). However, when the incident illumination is low, the photocurrent is small, causing the photodiode to discharge slowly. In this case, it would be necessary to increase the integration time in order to allow the photodiode to discharge enough to provide a detectable signal at V_{OUT}. Unfortunately, the integration time is constant for all the pixels in the frame. Therefore, if t_{int} is set to accommodate the bright light, the dim light regions in the scene will appear overly dark; similarly, if t_{int} is set to accommodate the dim light regions, the bright regions will appear overexposed.

In a PFM scheme, the minimum detectable illumination is the intensity that results in a photocurrent that will discharge the photodiode at the minimum rate specified by the system. The upper limit on dynamic range is bound by the reset delay, t_{reset}, which is the time required to turn on the reset transistor and recharge the photodiode.

Figure 8.6 plots photocurrent versus frequency of PFM pulses using the five comparators under consideration. Due to its fast switching speeds, the inverter operates much quicker than

the other comparators and it has a lower minimum detectable photocurrent; therefore the inverter has the potential of achieving a higher dynamic range.



I_{photo} [pA]	Two-Stage Comparator	Symmetrical OTA	Symmetrical OTA with Feedback	Schmitt Trigger	Inverter with Reduced V_{RST}
10^0	403	402	474	50	695
10^1	3,704	3,467	3,804	601	10,724
10^2	36,364	34,734	36,670	6,135	117,800
10^3	344,828	329,272	340,252	61,128	1,072,156
10^4	2,857,143	2,477,701	2,002,403	587,149	10,629,252
10^5	14,285,714	13,104,442	8,984,726	4,991,302	80,580,177
Trendline	$y = 464.71x^{0.9253}$	$y = 456.17x^{0.9173}$	$y = 568.22x^{0.872}$	$y = 56.977x^{0.999}$	$y = 922.97x^{1.0077}$
R^2	0.9972	0.9974	0.9949	0.9995	0.9978

Figure 8.6: Photocurrent versus Frequency of PFM pulses.

To determine the lower and upper bounds of dynamic range, let the frame rate be 30fps (a practical value); therefore, t_{int} = 33ms. If the minimum number of pulses that are required to resolve a valid signal is $pulses_{min}$, then the minimum frequency, $f_{min} = \frac{pulses_{min}}{t_{int}}$. Furthermore, let there be at least a 70% integration duty cycle; therefore, the maximum frequency, $f_{max} = \frac{0.3}{t_{reset,worstcase}}$, where $t_{reset,worstcase}$ is the highest reset delay from simulation. The optical dynamic range is the ratio between the photocurrent corresponding to the maximum frequency ($I_{photo@f_{max}}$) and the photocurrent corresponding to the minimum frequency ($I_{photo@f_{min}}$):

$$DR = \frac{I_{photo@f_{max}}}{I_{photo@f_{min}}} \tag{8.7}$$

Using reset delay values listed in Table 8.1 and the frequency versus photocurrent relationships from Figure 8.6, Table 8.2 lists the theoretical achievable dynamic ranges for each of the comparators.

Table 8.2: PFM Dynamic Range

	Symmmetrical OTA	Symmetrical OTA with Feedback	Two Stage Comparator	Schmitt Trigger	Inverter
$t_{reset,worstcase}$	17.38 ns	45.11 ns	86.57 ns	2.01 ns	1.74 ns
f_{max}	17,261,220 Hz	6,650,410 Hz	3,465,240 Hz	149,536,437 Hz	172,910,663 Hz
	$f = 464.71I^{0.9253}$	$f = 568.22I^{0.872}$	$f = 464.71I^{0.9253}$	$f = 56.977I^{0.999}$	$f = 922.97I^{1.0077}$
$I_{photo@f_{max}}$	97,875.38 pA	46,293.31 pA	15,317.38 pA	2,663,624.02 pA	170,743.87 pA
f_{min}	3030 Hz	3030 Hz	3030 Hz	3030 Hz	3030 Hz
$I_{photo@f_{min}}$	7.79 pA	6.74 pA	7.50 pA	52.86 pA	3.22 pA
Dynamic Range	**82 dB**	**77 dB**	**66 dB**	**94 dB**	**94 dB**

For reference, a typical dynamic range for a commercial CMOS monochrome sensor is approximately 50dB [34].

Power

Due to the constant switching of the photodiode voltage, the PFM scheme is inherently high in dynamic power [66]:

$$P_{dynamic} = C_L V_{DD}^2 f \tag{8.8}$$

However, the system-level power can be minimized by enabling the photo integration cycle only when a pixel's row is enabled. Therefore, for an mxn sensor array, the integration of a pixel would only occur 1/n of the total time. The photo integration process can be easily disabled by tying the ROW_SELECT signal to the logic controlling the reset transistor (e.g. Figure 8.1b).

Table 8.3 lists the average power during PFM pixel operation, with $I_{photo} = 0.1$mA. The load is a 3-bit static counter.

The inverter comparator consumed nearly twice the amount of power as the other comparators due to its high frequency. The other source of power consumption in the inverter is short circuit

Table 8.3: PFM Power Measurements

	Symmmetrical OTA	Symmetrical OTA with Feedback	Schmitt Trigger	Inverter
Frequency [Hz]	13,104,442	8,984,726	4,991,302	80,580,177
$P_{average}$ [μW]	69.00	67.85	73.81	128.32

current incurred during the switching of the inverter and both the pullup and pulldown transistors are briefly on. This is a tradeoff between performance (which results in high dynamic range) and power.

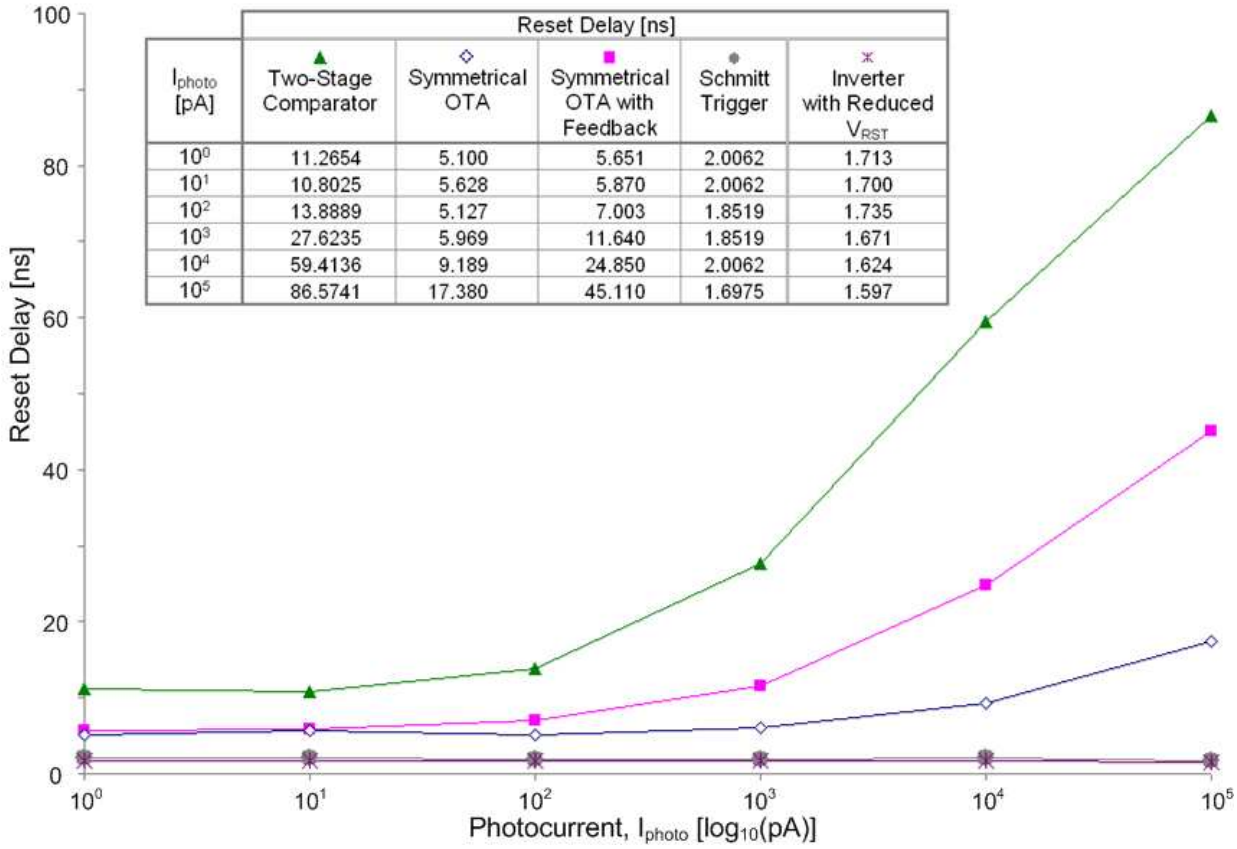
Linearity

Figure 8.7 shows that the pixels using the Schmitt trigger and inverter comparators exhibited the best linearity in response to the photocurrents. These two designs also had the most constant and lowest reset delays.

Other Issues to Consider/Resolve

In addition to dynamic range, power, and linearity, there are several issues that need to be resolved for a PFM pixel design, including:

- Non-uniformity between pixels (due to mismatch between comparators); and
- Coupled-noise (which might result from high frequency switching between neighbouring pixels).



I_{photo} [pA]	Two-Stage Comparator	Symmetrical OTA	Symmetrical OTA with Feedback	Schmitt Trigger	Inverter with Reduced V_{RST}
10^0	11.2654	5.100	5.651	2.0062	1.713
10^1	10.8025	5.628	5.870	2.0062	1.700
10^2	13.8889	5.127	7.003	1.8519	1.735
10^3	27.6235	5.969	11.640	1.8519	1.671
10^4	59.4136	9.189	24.850	2.0062	1.624
10^5	86.5741	17.380	45.110	1.6975	1.597

Figure 8.7: Photocurrent versus Reset Delay.

Chapter 9

Conclusions

This thesis presented a discussion on system-level and chip-level active vision designs: Part I reviewed the design and implementation of a saccadic camera prototype for active vision applications, and Part II presented custom CMOS image sensor designs considerations.

9.1 Saccadic Camera: Mike

Mike is a system-level implementation of an active vision system, where the wide-angle scene is captured in low-resolution, and salient regions are revisited by a high-resolution subwindow. The final product is a dual-resolution reconstruction of the scene where only the salient regions contain high resolution details. The camera's exposure time can also be locally adjusted for each foveal fixation, thus increasing the overall dynamic range of the salient regions in the scene.

The following lists the system requirements for a typical active vision application:

1. low processing time and power;
2. minimal data transmission;
3. real-time access to data presented in a useful form; and
4. fault tolerance against minor damage to potentially vulnerable system components.

9.1.1 Low Processing Time and Power

By performing bulk processing on the low-resolution image and attending only to salient regions, this system allocates resources efficiently and effectively. Measurements of algorithmic runtimes on various image resolutions show that, for the most computation intensity algorithm (corner detection), the execution time on the 1/4 SXGA resolution image requires 1/20 (5%) of the time to analyze the full resolution image. This results in major time and consequently, power, savings.

9.1.2 Minimal Data Transmission

Assuming an average of 100 salient regions captured during saccade mode, then 100x32x32 = 102,400 pixels contain high-resolution information. Out of the possible 1280x1024 pixels in the full SXGA resolution sensor, the top 100 salient regions constitute 7.8% of the sensor space. Therefore, over 92% of irrelevant data is filtered from the transmission stream, allowing for faster transmission and efficient use of the available bandwidth.

9.1.3 Real-time Access to Data Presented in a Useful Form

The output of Mike is high-resolution salient information captured at locally adjusted exposure times. This allows active vision applications immediate access to information relevant to their tasks, with irrelevant information removed. Although an alternate configuration of the Mike system has the potential for real-time performance, the current prototype does not perform quickly enough for real-time use. Combined feature extraction, on average, is achieved in under 2 seconds; this can be further reduced by optimizing the code for speed and by multi-threading the processing with camera commands. The major bottleneck of the system, however, is the camera unit: the system performance is constrained by the speed of reprogramming subwindow locations. This is a constraint of the particular camera model used in the prototype and does not hinder the potential of realizing a truly real-time Mike system with a different choice of hardware components, such as a custom imaging chip.

9.1.4 Fault Tolerance

The adaptation mode allows the Mike prototype to utilize its mechanical components to calibrate physical locations in the real world relative to its own self-motion. This allows the system to function even in the presence of lens distortion and provides a level of tolerance against certain hardware faults.

9.2 CMOS Image Sensor Design Considerations

CMOS image sensors are more suitable for computer vision applications than CCD cameras, due to the addressability of pixels and integrability of circuits within pixels. Mike's saliency map may be used to provide region of interest locations off-chip.

As technology scales, dynamic range becomes an important issue for image sensors, due to reduced voltage swing. An PFM readout scheme increases the dynamic range capabilities of CMOS pixels, because the scheme does not require large voltage swings. A PFM pixel was proposed using an inverter as the comparator. This pixel is able to achieve a theoretical dynamic range of 95dB.

9.3 Research Motivation Revisited

Figure 9.1a/c show the recorded eye movements of an observer examining the photographs of two female faces; Figure 9.1b/d show the respective saliency map output using those same photographs as input. As Yarbus noted, the human observer instinctively attends to the eyes, nose and lips; these are indeed the most expressive aspects of the face [3]. Nevertheless, even without an understanding of the word 'expressive', Mike, who is programmed to look for corners, contrast, and edges, is also drawn to the eyes. In terms of the corner detection algorithm, this makes perfect sense, as the shape of the eyes comprise sharp, distinct corners, and the striking contrast between the pupil and the iris also provides a stimulating demand for attention. While the emphasis of Mike's attention is on the eyes and outline of the face/hair, however, Mike does not attend much to the lips. This is where Mike's operation diverges from its biological inspiration. It should

be understood that the intention of the Mike system is to realize a practical design that meets the needs of an active vision system. While it borrows from certain aspects of the biological eye, it is not a model of the eye and not intended to mimic the eye's exact functions. There is, however, flexibility in Mike's programming; the weighting of the various feature detectors can be adjusted, and the addition of categorization rules in Mike programming can be used to train Mike to search for specific features or types of objects. Thus, Mike *could* be programmed to behave as its biological counterpart in the task of face examination.

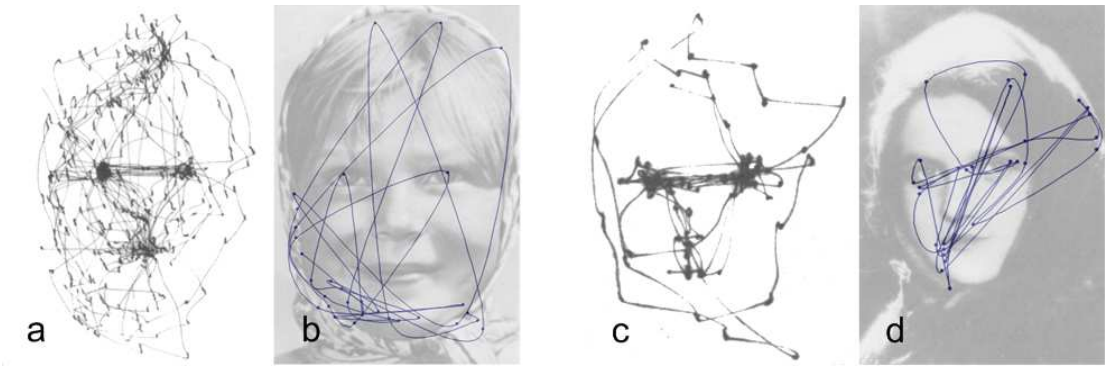


Figure 9.1: Comparison of recorded eye movements to simulation results. a and c show the recorded eye movements of an observer looking at photographs; b and d show the respective saliency map output using those same photographs as input.

Figure 9.2 shows an analysis of the salience of nine famous faces. In all cases, the saliency map counts the eye among the top salient regions in the photograph. This perhaps suggests that, even if we did not consciously consider eyes as important indicators of a person's mood, we would as likely fixate on them during a bottom-up search, as well as a top-down search. The nose and the lips, on the other hand, are generally ignored by the Mike code, suggesting that these two facial elements draw attention due to learned behaviour.

9.3.1 Practical Considerations and Recommendations for Future Works

Implementation and testing of the Mike prototype revealed the difficulties in realizing the design in hardware. The feature extraction routines generally performed much better on test images captured by commercial cameras (typically with ideal lighting conditions), rather than images

Figure 9.2: Saliency map results on analysis of famous faces [45],[8].

captured by the system camera. In a practical prototype, there exists a tradeoff between camera settings (i.e. exposure time and subwindow size) and the performance of the system.

A second generation prototype design of Mike should consider the use of either a commercial camera with faster subwindow programming and more flexible exposure time settings, or a custom CMOS imaging chip, perhaps with some on-chip preprocessing capabilities to generate the saliency map. A fast region of interest addressing scheme, such as the skip logic scheme [62], could be used to quickly access the foveal subwindows. Furthermore, the use of PFM pixels could increase the dynamic range of the system; if high dynamic range is unnecessary, the PFM scheme could also be used to achieve fast frame rates, as the reduced voltage swing requirements of PFM pixels means that integration time (i.e. exposure time), can be shortened.

9.3.2 Mike for the Real World

Although the previous section discusses the practical problems associated with the Mike prototype, the concept of a saccadic camera is nonetheless potentially useful for active vision applications. The main benefit of a saccadic camera is its selective attention towards salient regions of the scene, which drastically reduces processing, transmission and storage of image data. If a custom CMOS imaging chip is able to provide fast enough re-programming of foveal subwindows, then a truly real-time saccadic camera could certainly be realized and used in an active vision system.

Appendix A

Prototype Code (C++)

A.1 Corner Detection

```
/*********************************************************************************************
*
*   Function:   DetectCorner
*
*   Purpose:    To estimate the locations of the intersection of two sharp/abrupt changes
*               in intensity.
*
*   Algorithm:  The input is formed by an image, I and two parameters: tau (the threshold on
*               lambda2) and the linear size of a square window (neighbourhood), say 2N+1
*               pixels.
*
*               1. Compute the image gradient over the entire image I.
*               2. For each image point p:
*                   a) form the matrix C over (2N+1)*(2N+1) neighbourhood Q of p.
*                   b) compute lambda2, the smaller eigenvalue of C.
*                   c) if lambda2 > tau, save the coordinates of p into a list, L.
*               3. Sort L in decreasing order of lambda2.
*               4. Scanning the sorted list top to bottom: for each current point, p,
*                  delete all points appearing further on in the list which belong to the
*                  neighbourhood of p.
*
*               The output is a list of feature points for which lambda2 > tau and whose
*               neighbourhoods do not overlap.
*
*********************************************************************************************/
void CMike::DetectCorner(PU8 pInputImage, unsigned w, unsigned h) {
    unsigned i, j;
    double SumEx2, SumEy2, SumExEy;
    unsigned a, b;
    double Qx[9][9];
    double Qy[9][9];
    double Lambda2;
    double DetQuadratic;
```

```
Corner* pCnr;
POSITION CurrentPos, NextPos;

for (j = 0; j < h; j++)
{
    for (i = 0; i < w; i++)
    {
        for (b = 1; b <= 4; b++)
        {
            for (a = 1; a <= 4; a++)
            {
                Qx[4-a][4-b] = ((i >= a) && (j >= b)) ? PixelMatrix[i-a][j-b].Jx : 0;
                Qy[4-a][4-b] = ((i >= a) && (j >= b)) ? PixelMatrix[i-a][j-b].Jy : 0;
            }
            for (a = 0; a <= 4; a++)
            {
                Qx[4+a][4-b] = ((i <= (w-1-a)) && (j >= b)) ? PixelMatrix[i+a][j-b].Jx : 0;
                Qy[4+a][4-b] = ((i <= (w-1-a)) && (j >= b)) ? PixelMatrix[i+a][j-b].Jy : 0;
            }
        }

        for (b = 0; b <= 4; b++)
        {
            for (a = 1; a <= 4; a++)
            {
                Qx[4-a][4+b] = ((i >= a) && (j <= (h-1-b))) ? PixelMatrix[i-a][j+b].Jx : 0;
                Qy[4-a][4+b] = ((i >= a) && (j <= (h-1-b))) ? PixelMatrix[i-a][j+b].Jy : 0;
            }
            for (a = 0; a <= 4; a++)
            {
                Qx[4+a][4+b] = ((i <= (w-1-a)) && (j <= (h-1-b))) ? PixelMatrix[i+a][j+b].Jx : 0;
                Qy[4+a][4+b] = ((i <= (w-1-a)) && (j <= (h-1-b))) ? PixelMatrix[i+a][j+b].Jy : 0;
            }
        }

        SumEx2 = 0;
        SumEy2 = 0;
        SumExEy = 0;
        for (b = 0; b < 9; b++)
        {
            for (a = 0; a < 9; a++)
            {
                SumEx2 += Qx[a][b] * Qx[a][b];
                SumEy2 += Qy[a][b] * Qy[a][b];
                SumExEy += Qx[a][b] * Qy[a][b];
            }
        }

        if (((SumEx2*SumEy2)-(SumExEy*SumExEy)) > 0) // if matrix is positive definite
        {
            DetQuadratic = pow((SumEx2 + SumEy2),2.0) - (4*((SumEx2*SumEy2)-pow(SumExEy,2.0)));
        }
        else
        {
            DetQuadratic = -1.0;
        }
```

```
        if (DetQuadratic > 0)
        {
            Lambda2 = ((SumEx2 + SumEy2) - sqrt(DetQuadratic))/2;
        }
        else
        {
            Lambda2 = -1.0;
        }

        if (Lambda2 >= TAU)
        // check if corner is on a white pixel i.e. lies on the object
        // and not outside the object
        {
            CurrentPos = CornerList.GetHeadPosition();
            NextPos = CurrentPos;
            Corner *pTemp;
            pCnr = new Corner;
            pCnr->Pt.x = i;
            pCnr->Pt.y = j;
            pCnr->Lambda2 = Lambda2;

            if (CurrentPos != NULL)
            {
                do
                {
                    CurrentPos = NextPos;
                    if (CurrentPos == NULL)
                    {
                        break;
                    }
                    pTemp = (Corner*)CornerList.GetNext(NextPos);
                } while (pTemp->Lambda2 > Lambda2);
                //} while ((pTemp->Lambda2 > Lambda2) && (pos != NULL));

                if (CurrentPos != NULL)
                {
                    CurrentPos = CornerList.InsertBefore(CurrentPos,pCnr);
                }
                else
                {
                    CurrentPos = CornerList.AddTail(pCnr);
                }
            }
            else
            {
                CurrentPos = CornerList.AddTail(pCnr);
            }
        }
    }
}

{
    // to compensate for problems iterating to the tail of the list, I'm adding a dummy tail
    // which will be deleted at the end of this routine
    Corner *pTail = new Corner;
```

```
    pTail->Lambda2 = -1;
    pTail->Pt.x = -1;
    pTail->Pt.y = -1;
    CornerList.AddTail(pTail);
}

POSITION posC = CornerList.GetHeadPosition();
Corner *pCnrC = (Corner*)CornerList.GetHead();

do
{
    pCnrC = (Corner*)CornerList.GetNext(posC);
} while (posC != NULL);


pCnrC = (Corner*)CornerList.GetTail();

POSITION posA = CornerList.GetHeadPosition();
POSITION posB1;
POSITION posB2;
Corner *pCnrA;
Corner *pCnrB;
Corner *pTempCnr;
int NumIter = 0;

posB1 = posA;
//pCnrA = (Corner*)CornerList.GetAt(posA);
pCnrA = (Corner*)CornerList.GetNext(posA); // this line will return the head
pCnrB = (Corner*)CornerList.GetNext(posB1); // this line is necessary to increment B1

do
{
    NumIter++;
    //for (pCnrB = (Corner*)CornerList.GetNext(posB1); (posB2 = posB1) != NULL;)
    for (pCnrB = (Corner*)CornerList.GetNext(posB1); posB1 != NULL;)
    {
        if (((pCnrA->Pt.x-7) <= pCnrB->Pt.x) && (pCnrB->Pt.x <= (pCnrA->Pt.x+7)) &&
            ((pCnrA->Pt.y-7) <= pCnrB->Pt.y) && (pCnrB->Pt.y <= (pCnrA->Pt.y+7)))
        {
            pTempCnr = pCnrB;
            posB2 = CornerList.Find(pTempCnr);
            if (posB1 != NULL)
                pCnrB = (Corner*)CornerList.GetNext(posB1);
            CornerList.RemoveAt(posB2);
            delete pTempCnr;
        }
        else
        {
            if (posB1 != NULL)
                pCnrB = (Corner*)CornerList.GetNext(posB1);
        }
    }

    // need to iterate through list to get posA because when I remove nodes from the list
    // it seems that I need to reiterate through the list to get valid position values
    posA = CornerList.GetHeadPosition();
```

```
        for (int n=0; n <= NumIter; n++)
            pCnrA = (Corner*)CornerList.GetNext(posA);
        posB1 = posA;
        posC = CornerList.GetTailPosition();
        pCnrC = (Corner*)CornerList.GetTail();
    } while (posA != NULL);

    posC = CornerList.GetTailPosition();
    pCnrC = (Corner*)CornerList.GetTail();
    posC = CornerList.GetHeadPosition();
    pCnrC = (Corner*)CornerList.GetHead();
    do
    {
        pCnrC = (Corner*)CornerList.GetNext(posC);
    } while (posC != NULL);
    posA = CornerList.GetHeadPosition();
    pCnrA = (Corner*)CornerList.GetHead();
    do
    {
        if (*(pInputImage + pCnrA->Pt.x + pCnrA->Pt.y*w) == BLACK)
        {
                pTempCnr = pCnrA;
                posB2 = CornerList.Find(pTempCnr);
                if (posA != NULL)
                    pCnrA = (Corner*)CornerList.GetNext(posA);
                CornerList.RemoveAt(posB2);
                delete pTempCnr;
        }
        else
        {
            pCnrA = (Corner*)CornerList.GetNext(posA);
        }
    }while (posA != NULL);

    pCnrA = (Corner*)CornerList.RemoveTail();
    delete pCnrA;
}
```

A.2 Edge Detection

A.2.1 Morphological Operator

```
/*******************************************************************************************
*
*   Function:   Erode
*
*   Purpose:    Perform morphological erosion on binary image
*
*******************************************************************************************/
PU8 CMike::Erode(PU8 pBinaryImage8Bpp, unsigned w, unsigned h) {
    unsigned x, y;
    PU8 pErodedImage = new U8[w*h];
    signed Index;
    unsigned ErodedValue, PixCount;

    for (y = 0; y < h; y++)
    {
        for (x = 0; x < w; x++)
        {
            Index = x + (y*w);
            ErodedValue = *(pBinaryImage8Bpp + Index); // centre pixel
            PixCount = 1;

            if (x != 0) // not in leftmost column
            {
                ErodedValue = ErodedValue + *(pBinaryImage8Bpp + Index - 1); // left pixel
                PixCount++;
            }
            if (y != 0) // not in top row
            {
                ErodedValue = ErodedValue + *(pBinaryImage8Bpp + Index - w); // upper pixel
                PixCount++;
            }
            if (x != (w-1)) // not in rightmost column
            {
                ErodedValue = ErodedValue + *(pBinaryImage8Bpp + Index + 1); // right pixel
                PixCount++;
            }
            if (y != (h-1)) // not in bottom row
            {
                ErodedValue = ErodedValue + *(pBinaryImage8Bpp + Index + w); //lower pixel
                PixCount++;
            }
            ErodedValue = (unsigned)(ErodedValue / PixCount);
            ((PU8)pErodedImage)[Index] = (ErodedValue < WHITE) ? BLACK : WHITE;
        } // for x
    } // for y
    return pErodedImage;
}
```

```
/*******************************************************************************************
*
*   Function:   Dilate
*
*   Purpose:    Perform morphological dilation on binary image
*
*******************************************************************************************/
PU8 CMike::Dilate(PU8 pBinaryImage8Bpp, unsigned w, unsigned h) {
    unsigned x, y;
    PU8 pDilatedImage = new U8[w*h];
    signed Index;
    unsigned DilatedValue;

    for (y = 0; y < h; y++)
    {
        for (x = 0; x < w; x++)
        {
            Index = x + (y*w);

            DilatedValue = *(pBinaryImage8Bpp + Index); // centre pixel

            if (x != 0) // not in leftmost column
            {
                DilatedValue = DilatedValue + *(pBinaryImage8Bpp + Index - 1); // left pixel
            }

            if (y != 0) // not in top row
            {
                DilatedValue = DilatedValue + *(pBinaryImage8Bpp + Index - w); // upper pixel
            }

            if (x != (w-1)) // not in rightmost column
            {
                DilatedValue = DilatedValue + *(pBinaryImage8Bpp + Index + 1); // right pixel
            }

            if (y != (h-1)) // not in bottom row
            {
                DilatedValue = DilatedValue + *(pBinaryImage8Bpp + Index + w); //lower pixel
            }

            ((PU8)pDilatedImage)[Index] = (DilatedValue > BLACK) ? WHITE : BLACK;

        } // for x
    } // for y

    return pDilatedImage;
}
```

```
/*********************************************************************************************
*
*   Function:   DiffImage
*
*   Purpose:    Find the difference between two images with the same dimensions.
*               Returns Image1 - Image2
*
*********************************************************************************************/
PU8 CMike::DiffImage(PU8 pImage1, PU8 pImage2, unsigned w, unsigned
h) {
    PU8 pDiffedImage = new U8[w*h];
    unsigned x, y, Index;

    Index = 0;
    for (y = 0; y < h; y++)
    {
        for (x = 0; x < w; x++)
        {
            ((PU8)pDiffedImage)[Index] = ((PU8)pImage1)[Index] - ((PU8)pImage2)[Index];
            Index++;
        }
    }
    return pDiffedImage;
}
```

A.2.2 Canny Edge Detector

```
/*********************************************************************************************
*
*   Function:   CalcGradient
*
*   Purpose:    Compute the gradient matrices Jx and Jy
*
*
*********************************************************************************************/
void CMike::CalcGradient(PU8 pInputImage, unsigned w, unsigned h) {
    unsigned i, j, index;
    index = 0;
    for (j = 0; j < h; j++)
    {
        for (i = 0; i < w; i++)
        {
            PixelMatrix[i][j].value = ((PU8)pInputImage)[index++];
        }
    }

    // convolve rows with [1 0 -1]
    for (j = 0; j < h; j++)
    {
        for (i = 0; i < w; i++)
        {
            if ((i == 0) || (i == (w-1)))
            {
                PixelMatrix[i][j].Jx = 0;
            }
            else
            {
                PixelMatrix[i][j].Jx =  PixelMatrix[i-1][j].value - PixelMatrix[i+1][j].value;
            }
        }
    }

    // convolve columns with [1 0 -1]
    for (i = 0; i < w; i++)
    {
        for (j = 0; j < h; j++)
        {
            if ((j == 0) || (j == (h-1)))
            {
                PixelMatrix[i][j].Jy = 0;
            }
            else
            {
                PixelMatrix[i][j].Jy =  PixelMatrix[i][j-1].value - PixelMatrix[i][j+1].value;
            }
        }
    }

    return;
}
```

```
/*********************************************************************************************
*
*   Function:   CalcEdgeProp
*
*   Purpose:    Estimate edge strength and orientation (Canny edge detector)
*               Es(i, j) = sqrt(Jx^2(i,j) + Jx^2(i,j))
*               Eo(i, j) = arctan(Jy/Jx);
*               Note: the assumption is that CalcGradient has been called already
*               Note: i'm skipping the gaussian smoothing for the canny algorithm
*
*
*********************************************************************************************/
void CMike::CalcEdgeProp(unsigned w, unsigned h) {
    unsigned i, j;
    RawPix Pix;

    for (j = 0; j < h; j++)
    {
        for (i = 0; i < w; i++)
        {
            Pix = PixelMatrix[i][j];
            PixelMatrix[i][j].Es = sqrt((Pix.Jx*Pix.Jx) + (Pix.Jy*Pix.Jy));
            PixelMatrix[i][j].Eo = (Pix.Jx != 0) ? atan(Pix.Jy/Pix.Jx) : atan(Pix.Jy/0.0001);
        }
    }
}

/*********************************************************************************************
*
*   Function:   CannySuppress
*
*   Purpose:    Detect thinned edges from output of the canny enhancer
*
*
*********************************************************************************************/
PU8 CMike::CannySuppress(unsigned w, unsigned h) {
    unsigned i, j, index;
    int Dk;
    double Mask[3];
    PU8 pImage = new U8[w*h];
    double Dir;

    // quantise the orientation
    index = 0;
    for (j = 0; j < h; j++)
    {
        for (i = 0; i < w; i++)
        {
            Dir = PixelMatrix[i][j].Eo;
            if (sqrt(Dir*Dir) <= PI/8)
            {
                Dk = ZERO;
            }
            else
            {
                if (sqrt(Dir*Dir) >= PI/8*3)
```

```
                {
                    Dk = NINETY;
                }
                else
                {
                    if (Dir < 0)
                    {
                        Dk = ONETHIRTYFIVE;
                    }
                    else
                    {
                        Dk = FORTYFIVE;
                    }
                }
            }
            Mask[1] = PixelMatrix[i][j].Es;
            switch(Dk)
            {
                case NINETY :
                    Mask[0] = (j > 0) ? PixelMatrix[i][j-1].Es : 0;
                    Mask[2] = (j < (h-1)) ? PixelMatrix[i][j+1].Es : 0;
                    break;
                case FORTYFIVE :
                    Mask[0] = ((i<(w-1)) && (j>0)) ? PixelMatrix[i+1][j-1].Es : 0;
                    Mask[2] = ((i>0) && (j<(h-1))) ? PixelMatrix[i-1][j+1].Es : 0;
                    break;
                case ONETHIRTYFIVE :
                    Mask[0] = ((i>0) && (j>0)) ? PixelMatrix[i-1][j-1].Es : 0;
                    Mask[1] = ((i<(w-1)) && (j<(h-1))) ? PixelMatrix[i+1][j+1].Es : 0;
                    break;
                default : // ZERO
                    Mask[0] = (i > 0) ? PixelMatrix[i-1][j].Es : 0;
                    Mask[2] = (i < (w-1)) ? PixelMatrix[i+1][j].Es : 0;
                    break;
            }

            PixelMatrix[i][j].In = ((Mask[1] >= Mask[0]) && (Mask[1] >= Mask[2])) ?
                Mask[1] : 0;

            ((PU8)pImage)[index++] = (U8)PixelMatrix[i][j].In;
        }
    }
    return pImage;
}
```

A.3 Connectivity

```
/*********************************************************************************************
*
*   Function:   ConnectCorners
*
*   Purpose:    To draw imaginary lines between all detected corners for shape detection
*
*   Algorithm:  Find all lines connecting two corners that do not
*               cross over any background pixels. If the line crosses a background
*               pixel, it may or may not connect two corners that belong to the same
*               object. However, if the line only crosses foreground pixels, then
*               both corners are guaranteed to belong to the same
*               object.
*
*
*********************************************************************************************/
void CMike::ConnectCorners(PU8 pBinaryImage, unsigned w, unsigned h)
{
    Point A, B;
    float Slope, Intercept;
    unsigned x, y;
    unsigned xMax, xMin, yMax, yMin;

    unsigned n = CornerList.GetCount();
    pConnections = new bool[n*n];

    for (y = 0; y < n; y++)
    {
        for (x = 0; x < n; x++)
        {
            *(pConnections + x + (y*n)) = false;
        }
    }

    {
        // to compensate for problems iterating to the tail of the list, I'm adding a dummy tail
        // which will be deleted at the end of this routine
        Corner *pTail = new Corner;
        pTail->Lambda2 = -1;
        pTail->Pt.x = -1;
        pTail->Pt.y = -1;
        CornerList.AddTail(pTail);
    }

    POSITION posA = CornerList.GetHeadPosition();
    POSITION posB = posA;
    Corner *pCnrA, *pCnrB;
    bool SameShape = true;
    unsigned IterA, IterB;

    pCnrA = (Corner*)CornerList.GetNext(posA); // this line will return the head
    pCnrB = (Corner*)CornerList.GetNext(posB); // this line is necessary to increment B

    A.x = pCnrA->Pt.x;
    A.y = pCnrA->Pt.y;
```

```
    B.x = pCnrB->Pt.x;
    B.y = pCnrB->Pt.y;

    IterA = 0;
    do
    {
        IterB = IterA+1;
        for (pCnrB = (Corner*)CornerList.GetNext(posB); posB != NULL; pCnrB = (Corner*)CornerList.GetNext(posB))
        {
            SameShape = true;
            B.x = pCnrB->Pt.x;
            B.y = pCnrB->Pt.y;

            xMax = (A.x > B.x) ? A.x : B.x;
            xMin = (A.x > B.x) ? B.x : A.x;
            yMax = (A.y > B.y) ? A.y : B.y;
            yMin = (A.y > B.y) ? B.y : A.y;

            Slope = ((float)B.y - (float)A.y)/((float)B.x - (float)A.x);
            Intercept = A.y - (Slope*A.x);

            for (x = xMin+1; x < xMax; x++)
            {
                y = (unsigned)((Slope*x) + Intercept + 0.5);
                if ((y >= yMin) && (y <= yMax))
                {
                    //((PU8)pBinaryImage)[x+(y*w)] = 192;
                    if (((PU8)pBinaryImage)[x+(y*w)] == BLACK)
                    {
                        SameShape = false;
                    }
                }
            }

            *(pConnections + IterA + (IterB++ * n)) = SameShape;

        }

        pCnrA = (Corner*)CornerList.GetNext(posA);
        A.x = pCnrA->Pt.x;
        A.y = pCnrA->Pt.y;

        posB = posA;

        IterA++;
    } while (posA != NULL);

    return;
}
```

Appendix B

Simulation Code (MatLab)

B.1 Feature Detection

```
clear all;

default_tau = 30000;
corner_weight = 7;
intensity_weight = 5;
segment_weight = 1;
edge_weight = 3;

for TestNum = 1:10,

for ResolutionCase = 1:4,

      switch ResolutionCase
            case 1
                Resolution = '1280x1024';
        zone = 17;
            case 2
                Resolution = '320x256';
        zone = 7;
            case 3
                Resolution = '160x128';
        zone = 3;
            case 4
                Resolution = '80x64';
        zone = 3;
        end;

    tau = default_tau;

    [I, imgMap] = imread(imgFile);

    nRows = size(I,1);
    nCols = size(I,2);

    % calc gradient
```

```
Hx = [1 0 -1];
Hy = Hx';
Jx1 = convn(I,Hx);
Jx = Jx1(:, 1:nCols);
Jy1 = convn(I,Hy);
Jy = Jy1(1:nRows, :);

% calc lambda2
Jx_temp = zeros(4+nRows+4, 4+nCols+4);
Jx_temp(5:nRows+4,5:nCols+4) = Jx;
Jy_temp = zeros(4+nRows+4, 4+nCols+4);
Jy_temp(5:nRows+4,5:nCols+4) = Jy;
for y = 5:nRows+4,
    for x = 5:nCols+4,
        Qx = Jx_temp(y-4:y+4,x-4:x+4);
        Qy = Jy_temp(y-4:y+4,x-4:x+4);

        SumEx2 = sum(sum(Qx.^2));
        SumEy2 = sum(sum(Qy.^2));
        SumExEy = sum(sum(Qx.*Qy));
        if (((SumEx2*SumEy2)-(SumExEy^2)) > 0) % if matrix is positive definite
            DetQuadratic = ((SumEx2+SumEy2)^2) - (4*((SumEx2*SumEy2)-SumExEy^2));
            lambda(y-4,x-4) = ((SumEx2+SumEy2) - sqrt(DetQuadratic))/2;
        else
            DetQuadratic = -1.0;
            lambda(y-4,x-4) = -1.0;
        end;

        clear Qx;
        clear Qy;
        clear SumEx2;
        clear SumEy2;
        clear DetQuadratic;

    end;
end;

% remove corner results in the same neighbourhood
lambda_temp = zeros(zone+nRows+zone, zone+nCols+zone);
lambda_temp(zone+1:nRows+zone,zone+1:nCols+zone) = lambda;
for y = zone+1:nRows+zone,
    for x = zone+1:nCols+zone,
        mask = lambda_temp(y-zone:y+zone,x-zone:x+zone);
        maxVal = max(max(mask));
        if lambda_temp(y,x) ~= maxVal
            lambda_temp(y,x) = -1.0;
        end;
    end;
end;
lambda2 = lambda_temp(5:nRows+4,5:nCols+4);
corner = lambda2>tau;

% if there are more than 100 corners detected, reset threshold
    corner(1:10,1:10) = 0; % delete false corner at top of image
    while (size(nonzeros(corner),1)) > 100
        tau = tau + 2000;
```

```
        corner = lambda2>tau;
    end;

[I_circle, imgMap] = imread(circleFile);

circlesize = size(I_circle,1);
halfcir = double((circlesize-1)/2);

I_markcorner = uint8(ones(halfcir+nRows+halfcir, halfcir+nCols+halfcir)*255);
I_markcorner(halfcir+1:nRows+halfcir, halfcir+1:nCols+halfcir) = I;
corner_temp = zeros(halfcir+nRows+halfcir, halfcir+nCols+halfcir);
corner_temp(halfcir+1:nRows+halfcir, halfcir+1:nCols+halfcir) = corner;
for y =  halfcir+1 : nRows+halfcir,
    for x = halfcir+1 : nCols+halfcir,
        if corner_temp(y,x) == 1
            I_markcorner(y-halfcir:y+halfcir, x-halfcir:x+halfcir)
            = uint8(double(I_markcorner(y-halfcir:y+halfcir, x-halfcir:x+halfcir)).* double(I_circle));
        end;
    end;
end;
clear corner_temp;

lambda(1:10,1:10) = 0;
maxLambda = max(max(lambda));
I_corner = (lambda)/maxLambda*(corner_weight); %normalize corners

%intensity map
meanval = mean(mean(I));
I_intensity = double(abs(I - meanval));
%normalize and multiply by weight
I_intensity = (double(I_intensity > 32) .* I_intensity)/255*intensity_weight;

%segmentation
% Binarize the image
OtsuLevel = graythresh(I);
I_bin = im2bw(I, OtsuLevel);

I_cornertemp = lambda;
I_segment2 = bwlabel(I_bin,5);
I_segment = double(zeros(nRows,nCols));
for n = 1:5
    n_reverse = 6-n;
    [MaxCornerVal xCorner]= max(max(I_cornertemp));
    [MaxCornerVal yCorner] = max(max(I_cornertemp'));
    MaxLabel = I_segment2(yCorner,xCorner);
    for x = 1:nCols
        for y = 1:nRows
            if (I_segment2(y,x) == MaxLabel)
                I_segment(y,x) = n_reverse;
                I_segment2(y,x) = 0;
            end;
        end;
    end;
end;
%normalize and multiply by weight
I_segmentPlot = (I_segment/max(max(I_segment)))*segment_weight;
```

```
    EC1 = edge(I,'canny');
    EC1_max = max(max(EC1));
    %normalize and multiply by weight
    I_edge = double(EC1/EC1_max)*edge_weight;

    %final
    I_total = I_edge + I_segmentPlot + I_corner + I_intensity;


    fid2 = fopen(outFileSalient, 'w');
    for n = 1:100,
        [salientVal x]= max(max(I_total));
        [salientVal y] = max(max(I_total'));
        lowerY = max(1, (y-zone));
        upperY = min(nRows, (y+zone));
        lowerX = max(1, (x-zone));
        upperX = min(nCols, (x+zone));
        I_total(lowerY:upperY,lowerX:upperX) = 0; %remove maximums from neighbourhood
        fprintf(fid2, '%d %d\n',x,y);
        if salientVal == 0
            n = 100;
        end;
    end;
    fclose(fid2);

    %%%%%%
    % clear all intermediate variables
    %%%%%%


end; % for loop of resolutions


end; % for loop of test cases
```

B.2 Object Categorization

```
numShapes = 8;

countCorners = 1; % 0 = off, 1 = on;
findColour = 1;
findSize = 1;

for TestNum = 100,

for ResolutionCase = 2,

      switch ResolutionCase
            case 1
                Resolution = '1280x1024';
            case 2
                Resolution = '320x256';
            case 3
                Resolution = '160x128';
            case 4
                Resolution = '80x64';
        end;
    [I, imgMap] = imread(imgFile);
    nRows = size(I,1);
    nCols = size(I,2);
    numCorners = zeros(numShapes+1,1);
    whitePixels = zeros(numShapes+1,1);
    greyPixels = zeros(numShapes+1,1);
    diagonal = zeros(numShapes+1,1); % diagonal size of bounding box around a shape

%%%%%%%%%%%segment the image%%%%%%%%%%%%%%%
    OtsuLevel = graythresh(I)
    I_bin = im2bw(I, OtsuLevel);
    I_segment = bwlabel(I_bin,numShapes) + 1;
    % add one so that object numbers can match matrix indices

%%%%%%%%%%%find corners%%%%%%%%%%%%%%%%%%
if (countCorners == 1)
    SE = ones(3,3);
    I_erode = imerode(I_bin,SE);

    topLeftMask = [-1 -1 -1 -1 -1
                   -1 -1 -1 -1 -1
                   -1 -1  1  1  1
                   -1 -1  1  1  1
                   -1 -1  1  1  1];
    topLeft = bwhitmiss(I_erode,topLeftMask);

    topRightMask = [-1 -1 -1 -1 -1
                   -1 -1 -1 -1 -1
                    1  1  1 -1 -1
                    1  1  1 -1 -1
                    1  1  1 -1 -1];
    topRight = bwhitmiss(I_erode,topRightMask);

    botLeftMask = [-1 -1  1  1  1
```

```
                -1 -1  1  1  1
                -1 -1  1  1  1
                -1 -1 -1 -1 -1
                -1 -1 -1 -1 -1];
    botLeft = bwhitmiss(I_erode,botLeftMask);

    botRightMask = [ 1  1  1 -1 -1
                     1  1  1 -1 -1
                     1  1  1 -1 -1
                    -1 -1 -1 -1 -1
                    -1 -1 -1 -1 -1];
    botRight = bwhitmiss(I_erode,botRightMask);

    topTriMask = [-1 -1  0 -1 -1
                  -1  0  1  0 -1
                   0  1  1  1  0
                   0  1  1  1  0
                   1  1  1  1  1];
    topTri = bwhitmiss(I_erode,topTriMask);

    leftTriMask = [-1 -1  0  0  1
                   -1  0  0  1  1
                    0  1  1  1  1
                   -1  0  0  0  0
                   -1 -1 -1 -1 -1];
    leftTri = bwhitmiss(I_erode,leftTriMask);

    rightTriMask = [ 1  0  0 -1 -1
                     1  1  0  0 -1
                     1  1  1  1  0
                     0  0  0  0 -1
                    -1 -1 -1 -1 -1];
    rightTri = bwhitmiss(I_erode,rightTriMask);

    corner = double(topTri) + double(leftTri) + double(rightTri) + double(topLeft)
    + double(topRight) + double(botLeft) + double(botRight);

    for y = 1:nRows,
        for x = 1:nCols,
            if (corner(y,x) == 1)
                %if more than one hit for the same corner
                %keep top left hit
                corner(y-1:y+1,x-1:x+1) = zeros(3,3);
                corner(y,x) = 1;
                numCorners(I_segment(y,x)) = numCorners(I_segment(y,x)) + 1;
            end;
        end;
    end;

end; % if countCorners

%%%%%%%%%%%%%%%%%classify colour/intensity of shapes%%%%%%%%%%%%%%%
if (findColour == 1)
    I_1D = I(:,:,1);

    I_white = I_1D > 200;
```

```
    I_grey = (I_1D > 100) & (I_1D < 200);

    for i = 2:numShapes+1,
        temp = double((I_segment == i)) .* double(I_white);
        whitePixels(i) = size(nonzeros(temp),1);
        clear temp;

        temp = double((I_segment == i)) .* double(I_grey);
        greyPixels(i) = size(nonzeros(temp),1);
        clear temp;
    end;

end; % if findColour

%%%%%%%%%%%%%determine size of shapes%%%%%%%%%%%%%%%%%%%%
    maxY = zeros(numShapes+1,1);
    minY = ones(numShapes+1,1) * nRows;
    maxX = zeros(numShapes+1,1);
    minX = ones(numShapes+1,1) * nCols;

    for y = 1:nRows,
        for x = 1:nCols,
            for i = 2:(numShapes+1),
            % ignore case 1 because that's the background
                if (I_segment(y,x) == i)
                    if (y > maxY(i))
                        maxY(i) = y;
                    end;
                    if (y < minY(i))
                        minY(i) = y;
                    end;
                    if (x > maxX(i))
                        maxX(i) = x;
                    end;
                    if (x < minX(i))
                        minX(i) = x;
                    end;
                end;
            end;
        end;
    end;

if (findSize == 1)

%     diagonal = zeros(numShapes+1,1);
% diagonal size of bounding box around a shape
    for i = 2:(numShapes+1),
        diagonal(i) = sqrt((maxY(i)-minY(i))^2 + (maxX(i)-minX(i))^2);
    end;
end; % if findSize
 %%%%%%%%

end; end;


% %%%%%%%%%%%%% begin categorization%%%%%%%%%%%%%%%%%%%
```

```
% % let 0 = no
% %     1 = yes
% %    -1 = not enough information
% isSquare = -1; % 1 = square, 0 = triangle
% % isTriangle = -1; % since there are only two shapes,
% %                  % can also assume that if it's not
% %                  % a square, then it's a triangle
% isLarge = -1; % 1 = large, 0 = small
% isWhite = -1; % 1 = white, 0 = grey
% % isGrey = -1; % again, since there are only two colours,
% %              % can assume that if an object isn't white
% %              % then it's grey

isSquare = ones(numShapes+1,1) * -1; % initial to unknown state
isLarge = ones(numShapes+1,1) * -1; isGrey = ones(numShapes+1,1) *
-1;

typeI = ones(numShapes+1,1) * 3; % 1 = columnA, 2 = columnB, 3 = unknown
typeII = ones(numShapes+1,1) * 3; % 1 = columnA, 2 = columnB, 3 = unknown
typeIII = ones(numShapes+1,1) * 3; % 1 = columnA, 2 = columnB, 3 = unknown
typeIV = ones(numShapes+1,1) * 3; % 1 = columnA, 2 = columnB, 3 = unknown
typeV = ones(numShapes+1,1) * 3; % 1 = columnA, 2 = columnB, 3 = unknown
typeVI = ones(numShapes+1,1) * 3; % 1 = columnA, 2 = columnB, 3 = unknown

for i = 2:numShapes+1, % ignore background
    switch numCorners(i)
    case 3
        isSquare(i) = 0; % triangle
    case 4
        isSquare(i) = 1; % square
    otherwise
        isSquare(i) = -1; % unknown
    end;

    if (diagonal(i) > 60)
        isLarge(i) = 1; % large
    else
        if (diagonal(i) > 10)
            isLarge(i) = 0; % small
        else
            isLarge(i) = -1; % unknown
        end;
    end;

    if (whitePixels(i) > 0)
        if (greyPixels(i) > 0)
            isGrey(i) = -1; % if the shape was found to be both grey and white
        else
            isGrey(i) = 0;
        end;
    else
        if (greyPixels(i) > 0)
            isGrey(i) = 1;
        end;
    end;
```

```
%typeI
if (isGrey(i) == -1)
    typeI(i) = 3;
else
    if (isGrey(i) == 1)
        typeI(i) = 1;
    else
        typeI(i) = 2;
    end;
end;

%typeII
if ((isGrey(i) == -1) | (isSquare(i) == -1))
    typeII(i) = 3;
else
    if ((isGrey(i) == 1) & (isSquare(i) == 0))
    | ((isGrey(i) == 0) & (isSquare(i) == 1))
        typeII(i) = 1;
    else
        typeII(i) = 2;
    end;
end;

%typeIII
if ((isGrey(i) == -1) | (isSquare(i) == -1) | (isLarge(i) == -1))
    typeIII(i) = 3;
else
    if ((isGrey(i) == 1) & (isLarge(i) == 1))
    | ((isLarge(i) == 0) & (isSquare(i) == 0))
        typeIII(i) = 1;
    else
        typeIII(i) = 2;
    end;
end;

%type IV
if ((isGrey(i) == -1) | (isSquare(i) == -1) | (isLarge(i) == -1))
    typeIV(i) = 3;
else
    if ((isLarge(i) == 1) & (isSquare(i) == 0))
    | ((isGrey(i) == 1) & (isLarge(i) == 1))
    | ((isGrey(i) == 1) & (isSquare(i) == 0))
        typeIV(i) = 1;
    else
        typeIV(i) = 2;
    end;
end;

%type V
if ((isGrey(i) == -1) | (isSquare(i) == -1) | (isLarge(i) == -1))
    typeV(i) = 3;
else
    if ((isGrey(i) == 1) & (isSquare(i) == 0))
    | ((isGrey(i) == 1) & (isLarge(i) == 1))
    | ((isGrey(i) == 0) & (isLarge(i) == 0) & (isSquare(i) == 1))
        typeV(i) = 1;
```

```
        else
            typeV(i) = 2;
        end;
    end;

    %type VI
    if ((isGrey(i) == -1) | (isSquare(i) == -1) | (isLarge(i) == -1))
        typeVI(i) = 3;
    else
        if ((isGrey(i) == 1) & (isLarge(i) == 1) & (isSquare(i) == 0))
        | ((isGrey(i) == 0) & (isLarge(i) == 0) & (isSquare(i) == 0))
        | ((isGrey(i) == 1) & (isLarge(i) == 0) & (isSquare(i) == 1))
        | ((isGrey(i) == 0) & (isLarge(i) == 1) & (isSquare(i) == 1))
            typeVI(i) = 1;
        else
            typeVI(i) = 2;
        end;
    end;


end;


figure
imshow(I)
for i = 2:numShapes+1,
    x = (maxX(i) - minX(i))/2 + minX(i)-4;
    y = (maxY(i) - minY(i))/2 + minY(i)+2;
    switch typeI(i)
    case 1
        text(x,y,'A');
    case 2
        text(x,y,'B');
    otherwise
        text(x,y,'?');
    end;

end;
saveas(gcf, outFileTypeI, 'bmp')
```

Appendix C

Photodiode Model

$$\begin{aligned} I_{total} &= A \times J_{total} \\ &= A(J_{diff} + J_{drift}) \\ &= A\left[qF_o\left(\frac{1-e^{-\alpha W}}{1+\alpha L_p}\right) + qp_{no}\frac{D_p}{L_p}\right] \\ &= I_{photo} + I_{dark} \\ &= AqF_o\left(\frac{1-e^{-\alpha W}}{1+\alpha L_p}\right) + Aqp_{no}\frac{D_p}{L_p} \end{aligned} \tag{C.1}$$

$$F_o = \frac{I_{o,transmitted}}{E_{ph}} \tag{C.2}$$

$$I_{o,transmitted} = (1-R)I_o \tag{C.3}$$

$$E_{ph} = \frac{hc}{\lambda} \tag{C.4}$$

$$\begin{aligned} W &= \sqrt{\frac{2\varepsilon(N_a+N_d)(V_o-V_{applied}}{aN_aN_d}} \\ &= \sqrt{\frac{2\varepsilon(N_a+N_d)(V_o+V_{rbias}}{aN_aN_d}} \end{aligned} \tag{C.5}$$

$$\alpha = \left(\frac{84.732}{\lambda} - 76.417\right)^2 \tag{C.6}$$

$$L_p = \sqrt{D_p\tau_p} \tag{C.7}$$

$$D_p = \frac{kT\mu_p}{q} \tag{C.8}$$

$$p_{no} = \frac{n_i^2}{N_d} \tag{C.9}$$

$$\begin{aligned} C &= \varepsilon A \\ &= \frac{A}{\sqrt{V_o - V}} \sqrt{\frac{q\varepsilon(N_a N_d)}{2(Na+Nd)}} \end{aligned} \tag{C.10}$$

$$V_o = \left(\frac{kT}{q}\right) \ln\left(\frac{N_a N_d}{n_i^2}\right) \tag{C.11}$$

where,

I_{total} : total current	n_i : intrinsic carrier concentration
E_{ph} : photon energy	k : Boltzmann constant
I_o : incident illumination	T : temperature
F_o : photonic flux	q : electronic charge
R : reflectivity	C : capacitance
W : depletion width	ε : permittivity
α : absorption coefficient	N_d, N_a : ion concentration
L_p : diffusion length	p_{no} : equilibrium minority carrier concentration
D_p : diffusion coefficient	τ_p : carrier lifetime

Equations C.1-C.5 are derived in Hornsey's short course notes [56], Equation C.6 is an empirical relationship for the visible spectrum [67]. The rest of the equations are derived from concepts discussed in [68].

Bibliography

[1] R.L Gregory, *Eye and Brain*. New York: McGraw-Hill Book Company, 1966.

[2] G. Palmieri, G. Anna Oliva, and M. Scotto, "C.R.T. spot-follower device for eye-movement measurements," *Kybernetik*, vol. 8, pp. 23–30, January 1971.

[3] Alfred Yarbus, *Eye Movements and Vision*. New York: Plenum Press, 1967.

[4] WebMuseum, "WebMuseum: Gogh, vincent van: Self-portraits." `http://www.ibiblio.org/wm/paint/auth/gogh/self/`, 2006.

[5] Boston College, "Digital archive of art." `http://www.bc.edu/bc_org/avp/cas/fnart/`, 2006.

[6] Technische Universităt Darmstadt, "Runlength encoding." `http://www.kom.e-technik.tu-darmstadt.de`, 2006.

[7] R. Volpe, T. Estlin, S. Laubach, C. Olson, and J. Balaram, "Enhanced mars rover navigation techniques," *IEEE Conf. on Robotics and Automation*, pp. 926–931, 2000.

[8] Pixar, "Monsters, inc.." http://www.pixar.com/featurefilms/inc/, 2006.

[9] J. Peng, A. Srikaew, M. Wilkes, K. Kawamura, and A. Peters, "An active vision system for mobile robots," *Proc. IEEE International Conf. on Systems, Man, and Cybernetics*, vol. 2, pp. 1472–1477, 2000.

[10] I.D. Reid, D.W. Murray, and K.J. Bradshaw, "Towards active exploration of static and dynamic scene geometry," *Proc. IEEE International Conf. on Robotics and Automation*, vol. 1, pp. 718–723, 1994.

[11] N. Srinivasa and R. Sharma, "Execution of saccades for active vision using a neurocontroller," *IEEE Control Systems Magazine*, vol. 17, pp. 18–29, 1997.

[12] A. Mazour and S. King, "Design and development of human equivalent inspection system," *Canpolar East Inc.*, 2002.

[13] D.J. Stack, C. Bandera, C. Wrigley, and B. Pain, "A real-time reconfigurable foveal target acquisition and tracking system," *Proc. SPIE Conf. on Acquisition, Tracking, and Pointing XIII*, vol. 3692, pp. 300–310, 1999.

[14] D. Kragic and H.I. Christensen, "Biologically motivated visual servoing and grasping for real world tasks," *Proc. IEEE Conf. on Intelligent Robots and Systems*, pp. 3417–3422, 2003.

[15] J. Herve, "Hand/eye coordination: Role of the active observer," *Proc. IEEE Conf. on Pattern Recognition*, pp. 292–296, 1996.

[16] A. Hauck, G. Passig, T. Schenk, M. Sorg, and G. Farber, "On the performance of a biologically motivated visual control strategy for robotic hand-eye coordination," *Proc. IEEE on Intelligent Robots and Systems*, pp. 1626–1632, 2000.

[17] B. Bishop, S. Hutchinson, and M. Spong, "On the performance of state estimation for visual servo systems," *Proc. IEEE Conf. on Robotics and Automation*, vol. 1, pp. 168–173, 1994.

[18] Michael Land, *Animal Eyes*. UK: Oxford University Press, 2001.

[19] St. Luke's Clinic, "Eye anatomy." `http://www.stlukeseye.com/anatomy/`, 2006.

[20] M. Land, "Motion and vision: why animals move their eyes," *J. Comp Physiol A*, pp. 341–352, 1999.

[21] Health Vision, "Snellen eye chart." `http://vision.about.com/od/glossary/g/snellen.htm`, 2006.

[22] K. Cha, K.W. Horch, and R.A Normann, "Simulation of a phosphene field based visual prosthesis," *Proc of IEEE Conf on Systems, Man and Cybernetics*, 1990.

[23] K. Cha, K.W. Horch, and R.A Normann, "Mobility performance with a pixelized vision system," *Vision Research*, 1992.

[24] James T. Fulton, *Biological Vision: A 21st Century Tutorial*. British Columbia: Trafford Publishing, 2004.

[25] G. W. Larson, H. Rushmeier, and C. Piatko, "A visibility matching tone reproduction operator for high dynamic range scenes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, pp. 291–306, 1997.

[26] Vicki Bruce and Patrick R. Green, *Visual Perception: Physiology, Psychology and Ecology, 2nd Ed.* UK: Lawrence Erlbaum Associates, 1990.

[27] "Dictionary.com/saccade." `http://www.dictionary.com`, 2006.

[28] C. Koch and S. Ullman, "Shifts in selective visual attention: towards the underlying neural circuitry," *Human Neurobiology*, vol. 4, pp. 219–227, 1985.

[29] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, pp. 1254–1259, 1998.

[30] QImaging, "Firewire vs. usb 2.0." `http://www.qimaging.com`, 2006.

[31] Jet Propulsion Laboratory, "Microrover radios and antennas." `http://mpfwww.jpl.nasa.gov/rovercom/itworks.html`, 2006.

[32] A. Kiely and M. Klimesh, "The icer progressive wavelet image compressor," *IPN Progress Report*, 2003.

[33] James Janesick, *Scientific Charge Coupled Devices*. United States: SPIE Press, 2001.

[34] PixeLink, "Pl-a65x imaging module datasheet." `http://www.pixelink.com`, 2006.

[35] B. Wilburn, N. Joshi, V. Vaish, E-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy, "High performance imaging using large camera arrays," *ACM Transactions on Graphics*, vol. 24, pp. 765–776, 2005.

[36] Emanuele Trucco and Alessandro Verri, *Introductory Techniques for 3-D Computer Vision*. NJ: Prentice Hall, 1998.

[37] NASA, "Mars rover." `http://marsrover.nasa.gov/gallery/`, 2006.

[38] The MathWorks, *MatLab: The Language of Technical Computing*, version 7.0.1 r14 ed., September 2004.

[39] E. Ashari, "Adaptive real-time thresholding for hardware implementation," Master's thesis, University of Waterloo, 2004.

[40] Liu Jianzhuang, Li Wenqing, and Tian Yupeng, "Automatic thresholding of gray-level pictures using two-dimension otsu method," *Proceedings, International Conference on Circuits and Systems*, vol. 1, pp. 325–327, 1991.

[41] Roger N. Shepard, Carl I Hovland, and Herbert M. Jenkins, "Learning and memorization of classifications," *Psychological Monographs*, vol. 75, pp. 1–42, 1961.

[42] E. Rosch and C. Mervis, "Family resemblance: Studies in the internal structure of categories," *Cognitive Psychology*, pp. 573–605, 1975.

[43] B. Rehder and A. Hoffman, "Eyetracking and selective attention in category learning," *Proceedings, 25th Annual Cognitive Science Conference*, p. 2003.

[44] Lingyun Zhang and Garrison W. Cottrell, "A computational model which learns to selectively attend in category learning," *Proceedings of the 4th IEEE International Conference on Development and Learning*, 2005.

[45] "Photo gallery." `http://www.holleywood.com`, 2006.

[46] "Eggert electronics." `http://www.eggertelectronics.com/SSC1BPCB.jpg`, 2006.

[47] Jakobson, "Trajectories of reaches to prismatically-displaced targets: evidence for 'automatic' visuomotor recalibration," *Experimental Brain Research*, pp. 575–89, 1989.

[48] J.P. David and M. Cohen, "Radiation-induced dark current in CMOS active pixel sensors," *IEEE Transactions on Nuclear Sciences*, 2000.

[49] G. Hopkinson, "Radiation effects in a CMOS active pixel sensor," *IEEE Transactions on Nuclear Sciences*, 2000.

[50] H. Mebrahtu, W. Gao, P. Thomas, W. Kieser, and R. Hornsey, "Heavy ion radiation damage simulations for CMOS image sensors," *Proc. of SPIE Conf. Photonics North*, 2004.

[51] M.T. El-Melegy and A.A. Farag, "Nonmetric lens distortion calibration: closed-form solutions, robust estimation and model selection," *IEEE Conference on Computer Vision*, 2003.

[52] L. Lucchese and S.K. Mitra, "Correction of geometric lens distortion through image warping," *Proceedings, International Symposium on Image and Signal Processing and Analysis*, 2003.

[53] D. Claus and A.W. Fitzgibbon, "A rational function lens distortion model for general cameras," *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[54] E. Fossum, "CMOS image sensors," *IEEE Transactions on Electron Devices*, vol. 44, pp. 1689–1698, 1997.

[55] A. El Gamal, "Trends in CMOS image sensor technology and design," *IEEE IEDM Technical Digest*, pp. 805–808, 2002.

[56] R. Hornsey, "Short course on image sensors." `http://www.cs.yorku.ca/visor`, 1999.

[57] X. Wang, "A high dynamic range CMOS image sensor with in-pixel light-to-frequency conversion," Master's thesis, University of Waterloo, 2004.

[58] A. El Gamal, "Ee 392b:image sensors, lectures posted online." `http://eeclass.stanford.edu/ee392b/`, 2005.

[59] Paul P. Lee, Robert M. Guidash, Teh-Hsuang Lee, and Eric G. Stevens, "Active pixel sensor integrated with a pinned photodiode," *United States Patent 5625210*, 1995.

[60] R. Burns, "Improved techniques for object location with CMOS image sensors," Master's thesis, University of Waterloo, 2003.

[61] F. De Nisi, F. Comper, L. Gonzo, M. Gottardi, D. Stoppa, A. Simoni, and J.A. Beraldin, "A CMOS sensor optimized for laser spot-position detection," *IEEE Sensors Journal*, vol. 5, pp. 1296 – 1304, 2005.

[62] O. Schrey, J. Huppertz, G. Filimonovic, A. Bussmann, W. Brockherde, and B.J Hosticka, "A 1 kx1 k high dynamic range CMOS image sensor with on-chip programmable region-of-interest readout," *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 911 – 915, 2002.

[63] J. Rabaey, *Digital Integrated Circuits A Design Perspective 1st Ed.* United States: Prentice Hall Electronics, 1996.

[64] A. Sedra and K. Smith, *Microelectronic Circuits, 4th Ed.* New York: Oxford University Press, 1998.

[65] J. Segura, J.L. Rossello, J. Morra, and H. Sigg, "A variable threshold voltage inverter for CMOS programmable logic circuits," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 1262–1265, 1998.

[66] A. Bellaouar and M. Elmasry, *Low-Power Digital VLSI Design Circuits and Systems.* United States: Kluwer Academic Publishers, 1995.

[67] M. Karray, P. Desgreys, and J. Charlot, "A CMOS inverter TIA modeling with VHDL-AMS," *Proc. of IEEE Conf. on System-on-Chip for Real-Time Applications*, 2003.

[68] S.M. Sze, *Physics of Semiconductior Devices 2nd Ed.* United States: John Wiley & Sons, 1981.