COMPUTER-AIDED INVARIANT FEATURE SELECTION

BY

ANTOIN LENARD BAKER

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENTS OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2008

ABSTRACT

Computer Vision is a topic composed of a wealth of research. Several of these topics include image restoration, scene reconstruction, and motion estimation. Another classical problem in computer vision includes determining whether or not an image contains a specific object; this branch of computer vision is called object recognition.

Object recognition (classification) is a broad topic that combines the expertise from several disciplines such as Machine Learning, Cognitive Psychology, Signal Processing, Physics, Mathematics, Information Theory, and Image Processing. However, it is well known that even the best object classification algorithms will produce poor results when given poor features to track. "Garbage In-Garbage Out" is the phrase coined by George Fuechsel of International Business Machines (IBM) to describe this phenomenon [1].

This proposal presents a method for reducing the load on the user in the feature selection process. By placing a computer "in the loop" of the feature selection process, the amount of time selecting appropriate features for object classification can be significantly reduced.

The goal is simple. Given at least two images (frames) containing the object of interest, the user simply selects the desired object in both images. Using a preprogrammed feature set, the computer will inform the user which features are best for recognizing the object of interest in future images.

To accomplish this objective, an object-oriented collection of possible features to track will be created (e.g. color, texture, intensity, centroid, moment of inertia). These features will be standalone objects (classes) with the same interface to allow for easy future expansion of the feature

set. Once the user selects the desired object in at least two frames, each feature class calculates the "distance" between the two selected objects. Features with relatively small "distance" between the two selected objects will be considered good features to track, and will be presented to the user. Based on expert knowledge, the user can simply accept or reject the proposed feature set.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1 – INTRODUCTION

The field of object detection and classification is a topic that requires expertise from a broad audience. Many disciplines have long been interested in human vision (Cognitive Psychology, Physics, and Medicine) or teaching machines to see like humans (Electrical Engineering, Computer Science). Although these practitioners make up the bulk of the vision/computer vision community, anyone who has an application where extracting information from images is important can automatically become a computer vision developer.

Cognitive Psychology is the field of Psychology that examines internal mental processes. The branch of Cognitive Psychology deals highly with the scientific method, as opposed to other areas of psychology, which deal with introspection (self observation).

The vast majority of Cognitive Psychology research deals with sensation and perception. Cognitive Psychologists are interested in how humans acquire information (Psychophysics), and organize information (Gestalt Psychology). Before a machine can be taught to see, it is first necessary to take a look at how humans handle perception. Cognitive Psychology is a relatively new field and has only officially been around since the 1960's. Some fifty years later, Psychologists and Medical Practitioners still do not fully understand how humans handle vision

The field of physics is also concerned with the computer vision problem. Whereas Cognitive Psychologists are interested in human perception and computer scientists are concerned with teaching machines to make inferences (decisions) from images; physicists are usually concerned with the image formation process. For example, physicists are concerned with the process of how light refracts when passed through a lens. By understanding the physics of image formation,

problems in image formation can be corrected though pre-processing. Regarding vision, Psychophysicists are concerned with image formation in human vision. Psychophysicists are faced with a daunting task because the human vision system is essentially a massively parallel black box. Moreover, standardized test results can vary by person based on that individual's prior experience, medical condition, blood sugar level, amount of sleep gained the night before, or simply how the question was worded! Psychophysicists have methods for eliminating human bias to an extent, but the research is still in its early stages.

Computer science and Electrical Engineering are the two professional fields of study that have taken on the task of machine perception. However, anyone with a personal computer can do research on their own. Only until relatively recent times have computers become powerful enough to run algorithms on decent size images. Now research that has long been restricted to advanced labs is now available to anyone with access to a computer.

Computer vision formally became a topic of study in the 1970's when computers became able to handle the large amounts of data that are inherent with image processing. Moore's law has been a blessing and a curse when pertaining to computer vision. Although the massive increase in computer power has facilitated the increase in computer vision research, this research is now available to everyone. Because computer vision is a very immature field, there are massive amounts of research being conducted without any unifying source. As of today, a formal computer vision problem has yet to be defined.

Because computer vision is such a vast field, the research has been split into several sub-fields, including object recognition and classification. Even though object detection and

classification is a subtopic of computer vision, the amount of research being conducted is still overwhelming.

Pertaining to computer vision, object detection can be defined as determining whether an image contains a specific object. Object Classification is slightly different. Classification assumes that we already have an object, and we are determining its state of nature (class). Despite this minor difference, object detection, recognition, and classification will be used interchangeably throughout this dissertation.

Because of the wealth of literature concerning object classification, an engineer or scientist faced with the task of determining the correct features and algorithms for object detection is faced with a very time consuming process. There are thousands of available features and hundreds of algorithms available for object classification. Experience helps when determining which features and algorithms to use; but because of the lack of unification in the computer vision community, this experience rarely gets passed from journeyman to apprentice. With so many different people working on the same project, information is bound to get lost if no steps are taken.

However, the Computer Science community has already solved this problem starting in the 1990's with the widespread adaptation of object-oriented programming. Object-oriented programming simply means thinking of your program as a set of objects that can interact with each other. This programming philosophy can be extended to the field of object detection.

There are simply too many available features and algorithms for one person to be proficient at all of them: however, by using an object-oriented programming philosophy, a person doesn't have to be an expert at "everything." Each feature that can be used to identify or classify an object will be made into its own class. All of these classes will have the exact same interface to the rest of

the program. As more features become available, the user can simply insert the new features into the program with very little effort. This database of features can continue to grow over time, without adding significant complexity to the original program.

Once a database of features has been created, the task of finding the best features to detect an object becomes simple. Instead of scouring massive amounts of literature for the "best" features to find the object of interest, the user simply selects the object in a minimum of two separate images and the computer tells the user which features are best for finding the object in future images.

By organizing all the features into a single place, valuable experience is preserved. When a person graduates or leaves a company, he/she doesn't take all of his/her knowledge with him/her. Moreover, with this approach, everyone will be able to contribute relatively seamlessly to the overall goal of accurate object classification without needlessly adding complexity to the system.

This chapter explained why an object-oriented approach would be beneficial in the field of object detection. The following chapters will give some background into the field of human and computer vision systems. Finally, the approach used for finding the best features for object detection will be discussed.

CHAPTER 2 – BACKGROUND

Ever since the early twentieth century, humans have been fascinated with the idea of teaching robots to imitate human behavior. In the 1920's, Karel Capek's play "R.U.R." was the first piece of entertainment to put the word "robot" into mainstream circulation [2]. Since the introduction of robots in the 1920's, a myriad of science fiction novels, plays, and movies have come about, showing robots that completely mimic and interact with humans. Robots have been portrayed as very friendly such as "Johnny Five" from the movie "Short Circuit", or on a one-way track for the destruction of humanity such as the "Terminator" series [3].

One of the great pioneers of robotic science literature is the famous Isaac Asimov. Asimov was a Russian Biochemist who wrote hundreds of science-fiction novels. Asimov's three laws of Robotics paved the way for the majority of future robotic fiction. Using Asimov's laws as a basis, science fiction writers have created robots that range from wife replacements [4], to robots that fight crime so that humans can stay out of harm's way [5].

One of the major assumptions in most of these movies is the robots ability to have vision that is equal, if not superior to that of a human's vision system. Since the 1920's, there have been vast improvements in technology. With the invention of the "modern" computer in the 1950's, very sophisticated labs were the only places one could find enough computational power to perform the most basic computer vision research. In 1965, Gordon Moore stated that computing power would double every two years [6], as shown in **Figure 1.**

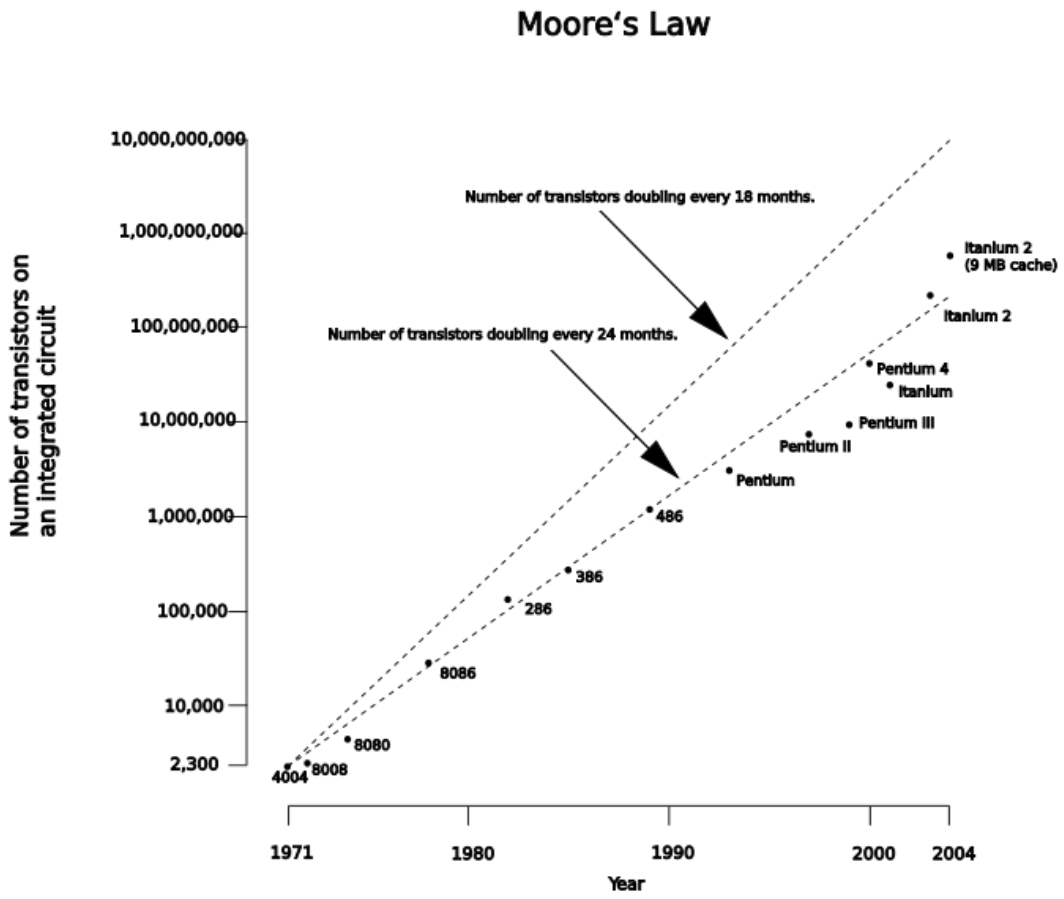

Figure 1: Moore's Law

Computing power has followed Moore's law very closely. In the year 2008, PDA's, gaming consoles and personal computers have more processing power than the mainframes of the 1970's [7]. With this exponential increase in processing power, computer vision research has become a possibility to anyone with access to a relatively modern personal computer. Because of this widespread availability of computing power, enormous research has been dedicated to the broad field of computer vision.

However, despite the vast amount of research being conducted in computer vision, humans are not even close to teaching machines to mimic human vision. Current computer vision research is focused on basic face detection and image segmentation; tasks most humans can do at birth. For example, newborn babies are able to recognize a person's mood from the person's facial expressions [8]. This fact is not a putdown on human ingenuity, but rather a testament to the creativity and complexity of a vision system so remarkable, that it must have been created by a higher being!

Because the overall goal of computer vision is teaching machines to imitate human vision and perception, additional attention will be given to the human vision system. The human vision system is extremely complex and many components have been "black boxed": but it only seems correct to investigate the subject that is to be mimicked. The following chapter will give an introduction to the human vision system. The topics to be covered in the next chapter range from low-level topics such as image formation, to high-level topics such as optical illusions. A full understanding of the next chapter is not needed to understand the goal of this research; however it gives the reader a background into the systems researchers are trying to reproduce.

CHAPTER 3 – THE HUMAN VISION SYSTEM

This chapter discusses how object classification occurs in humans. First, it gives a brief overview of the low level topics of human vision including the structure of the human eye and image formation. Second, it gives a discussion of the higher-level visual properties of perception. Finally, the chapter discusses some of the research being conducted in the field of human visual perception.

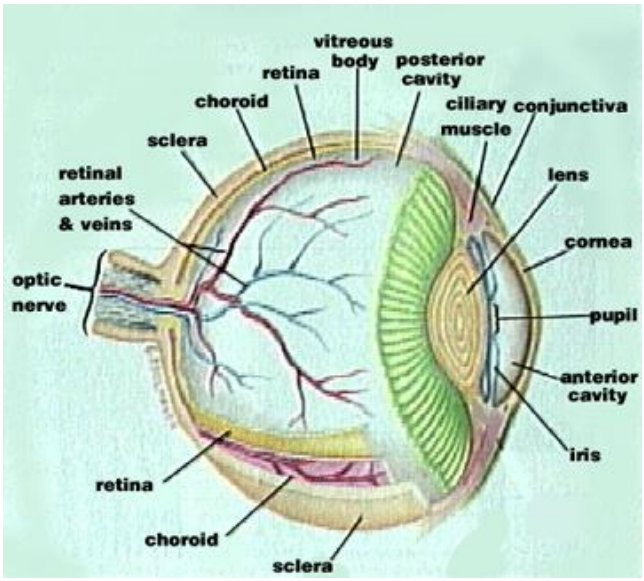

Figure 2: The Human Eye (www.cas.vanderbilt.edu/bsci111b/eye/human-eye.jpg)

The human eye (**Figure 2**) can be thought of as a 22-millimeter camera with an extremely high resolution. There are several studies that suggest a camera must have a resolution of 550-600 megapixels in order to match the resolution present in the human eye [9]! Other studies have shown that the human eye behaves more like a contrast detector, rather than an absolute detector such as a Charged-Couple Device (CCD) camera. These studies have proven the human eye to have a contrast ration of 1,000,000:1, or higher [10].

The human eye is an electromagnetic sensor that is sensitive to electromagnetic radiation between the wavelengths of approximately 370-730 nanometers. Light enters the eye through the cornea, which provides fixed focusing power very similar to a camera. Further focusing is accomplished by the lens, which changes its shape based on the distance between the fixated object and the observer. Most modern digital cameras perform autofocus by maximizing the contrast (value of derivatives) in the image. However, the image must have objects that contrast each other or the maximization algorithm will not work: for example, a digital camera will not be able to focus on a blank wall [11].

The pupil controls the intensity of the image and the lens controls the focus of the image. After the image passes through these filters, the resulting image will be inverted and flipped on the retina (**figure 3)**. However, the human brain has evolved to take care of this problem.

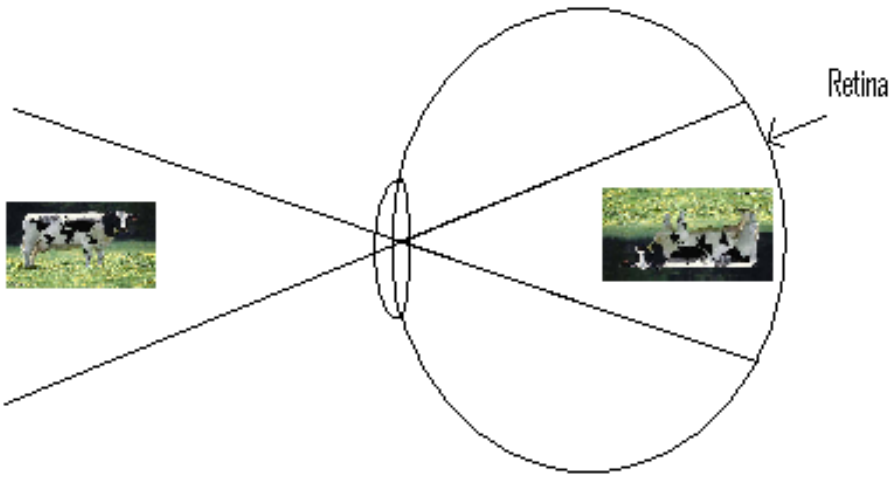

Figure 3: Inversion of Image on Retina

When the image lands on the retina, two types of receptors detect it. These two photoreceptors in the eye are called “rods” and “cones”. Rods are responsible for vision at low light levels and have very poor spatial acuity. The cones are the exact opposite. Cones respond at high light levels and have very accurate spatial acuity. There are three different types of cones in the eye, which respond to different wavelengths of light (red, green, and blue) [12]. Once the image is formed on the retina, the resulting image is carried from the eye to the brain by the optic nerve. The pathway from the eye to the brain is a complicated one, but knowledge of these details is outside the scope of this chapter. As the image passes from the eye to the brain, the visual process changes from sensation (image detection) to one of perception (image understanding).

While the details of image formation in the human eye may be of great concern to biologists or other medical professionals, image understanding is where the fun begins for the computer vision engineer. With this in mind, the rest of this chapter will be dedicated to the higher-level processes of how humans interpret and understand scenes once they have been formed by the lower-level systems.

As light passes through the cornea, lens, and pupil, it forms patches of light on the retina. However humans rarely see patches of light (unless under the influence of some mind-altering drug), rather we see an organized world consisting of different objects interacting with each other based on a perceptual set. The process is so natural that it is unperceived to most humans until an event occurs that violates our previous knowledge of the world. For example, the following two figures (**Figures 4 & 5)** illustrate this phenomenon**.** These are two images that violate most people’s perceptual organization experience, and therefore cause great confusion when encountered.

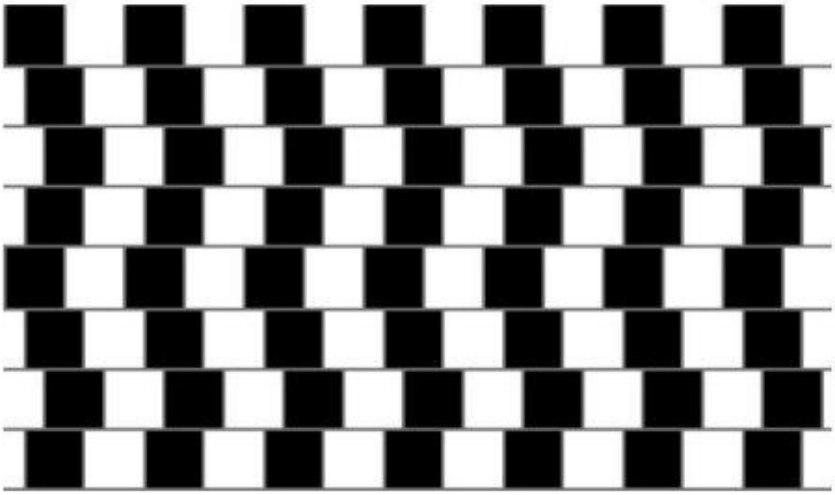

Figure 4: The Horizontal Lines Are Actually Parallel (http://kids.niehs.nih.gov/illusion/illusions3.htm)

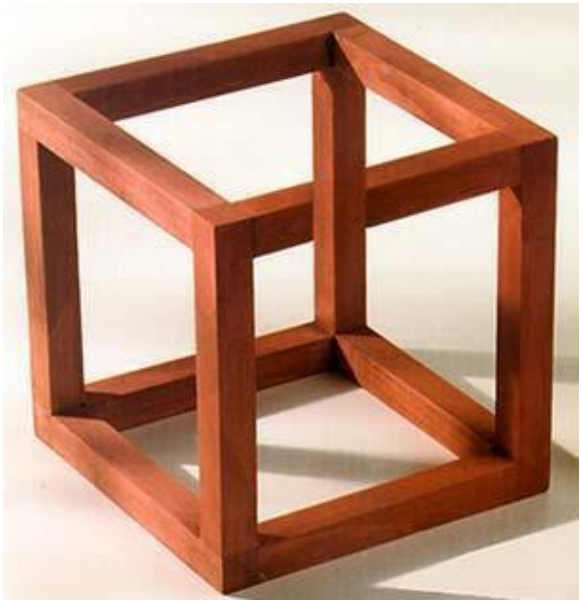

Figure 5: The Impossible Object (http://kids.niehs.nih.gov/illusion/illusions20.htm)

The Optical Illusion field is one of cognitive psychology that focuses on how the brain extracts meaningful information from raw data. The principle driving force behind perceptual organization is Gestalt psychology. The overall theme of Gestalt psychology is that "The whole is more than the sum of the parts [13]." According to Gestalt psychologists, the main idea behind their research is that the brain will produce the simplest possible organization allowed by the conditions [14]. For example, most humans will see the image in **Figure 6** as three overlapping disks. The image could have been interpreted as three balls place in front of each other, but the simplest interpretation states otherwise [15].

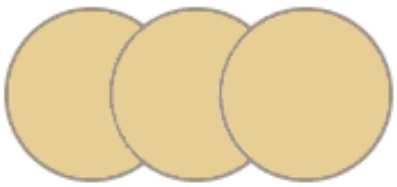

Figure 6: Gestalt Simplicity Principle
(http://encarta.msn.com/encyclopedia_761571997_2/Perception_(psychology).html)

Gestalt psychologists have long been studying how humans perform the act of object recognition and classification. The first step in recognizing an object is separating that object from its background. Gestalt psychologists believe humans accomplish these actions based on six main types of grouping principles: 1.Proximity 2.Similarity 3.Continuity 4.Closure 5.Common-Region 6.Connectedness (see Figures 7 through 12).

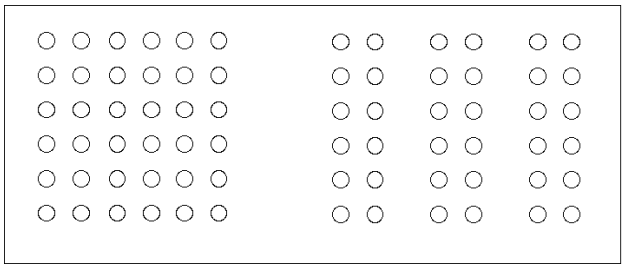

Figure 7: Gestalt Principle of Proximity

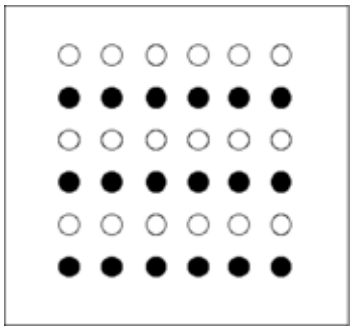

Figure 8: Gestalt Principle of Similarity

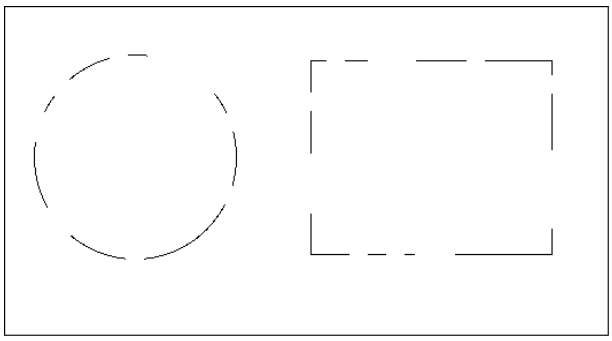

Figure 9: Gestalt Principle of Closure

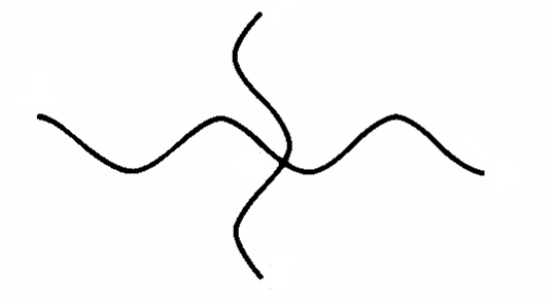

Figure 10: Gestalt Principle of Continuity

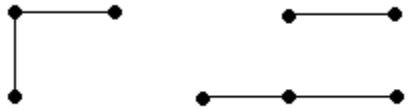

Figure 11: Gestalt Principle of Connectedness

Figure 12: Gestalt Principle of Common Region

Proximity simply implies that elements that are close together will tend to be grouped together. The principle of *similarity* says that elements similar in appearance will tend to be grouped together. *Continuity* says humans will organize elements along continuous contours. *Closure* shows that display elements that make up a closed figure will tend to be grouped together. *Connected* shows that objects that are somehow visually connected will be grouped together. *Common-region* or *common fate* implies that elements moving in the same direction or moving similarly will tend to be grouped together. In computer vision, the research of grouping related objects falls under the topic of image segmentation.

Once the image has been grouped into similar elements according to the above grouping principles, the human process of object recognition and classification begins. Although studying the human vision system can greatly aid in teaching computers to segment images, the same cannot be said about teaching computers to recognize objects. This problem occurs because humans and computers perform pattern recognition in a fundamentally different manner. Whereas humans utilize a top-down method (recognizing the object as a whole), computers use a bottom-up approach, meaning that they search for a collection of "features" to recognize an object.

This top-down approach can be observed by the Gestalt properties of emergence, reification, and invariance. Emergence means that the detected object is identified all at once, instead of a set of certain features. For example, most humans can recognize the dog in **Figure 13** even though it is completely disconnected and blends with the background. Reification implies that humans add more information than what is present in the original scene. Looking at **Figure 14** one can observe the triangle and the three-dimensional object even though no such objects exist. Invariance implies that humans are able to recognize object even if the features are completely distorted.

Figure 13: Gestalt Principle of Emergence

Figure 14: Gestalt Property of Reification

Gestalt psychology gives a description of what occurs in human vision, but it doesn't necessary explain how these processes occur. Because of the holistic nature of the human vision system, traditional pattern recognition techniques are virtually impossible. The initial step in pattern recognition is the process of feature analysis, but the human vision system is essentially feature invariant, making most pattern recognition techniques useless. Most humans are able to recognize a particular car whether it's night or day, whether the car is dirty or clean, whether the car is missing tires, upside-down, or has even been in an accident. It will be shown in the *Methodology* section of this dissertation how the holistic nature of the human vision system will be taken into account.

One area of pattern recognition that is used to make measurements of the human vision system is Signal Detection Theory. Signal detection theory is a method to quantify the ability to discern between a signal and noise. In the beginning, signal detection theory was used extensively by radar researchers [16]. Green and Swets later introduced signal detection theory into the field of psychophysics in 1966 [17]. Signal detection theory is used when psychologists want to measure how humans make decisions under conditions of uncertainty. Signal detection theory assumes that the noise and the signal are unchangeable, and that the variations in measurements are due to the active decision making of the human.

For example, if researchers want to determine a human's ability to recognize an object against a background, they present him with a large number of mixed images. The human classifies these images in to "object present" or "object absent." The researchers then measure the number of *hits* and *false alarms*. Afterwards, the researchers are able to use a lookup table to measure the human's sensitivity and his response bias. Experimenters use two main visualization techniques in Signal Detection Theory.

The first method is the confusion matrix (**Figure 15**). The confusion matrix is more like a matching matrix that simply graphs the number of *hits, misses, false alarms, and correct rejections.* The second method for visualizing Signal Detection Information is the Receiver Operating Characteristic (ROC). The ROC is a plot of the hit rate versus the false alarm rate for a sensor as its discrimination threshold is varied (**Figure 16**). For example, let's assume there is a pile of toothpicks on a table varying from 1 cm to 10 cm. The human is instructed to pick out all the toothpicks in the range of 1.5 cm to 9 cm. The number of correct identifications (hits) and false identifications (false alarms) is then plotted on the ROC curve as a point. The human is then instructed to pick out all the toothpicks that vary in range from 3 cm – 7cm, then 5cm – 6cm and so on. As the selection criterion gets tougher, the number of false alarms goes up and the number of hits goes down. The resulting curve from plotting these points will form a figure such as the **Figure 16**. A ROC curve used with real data will generally not be as smooth as the results shown in the figure, but the curve will follow the same general pattern.

While Signal Detection Theory may give the researcher a method of determining how well a human's vision system can perform a particular task, it still doesn't explain how the actual process occurs. Much more research is needed in the field of human object detection before computers can begin to mimic the actual process. Humans have only begin to scratch the surface of the human vision system, and in due time humans will gain a more complete understanding of how the process works.

Nevertheless, several exciting computer vision projects are currently being conducted without a full knowledge of the human vision system. The next chapter will give an introduction to some computer vision fundamentals and will discuss current computer vision research topics with a focus on object detection.

	Actual Positive	Actual Negative
Predicted Positive	"Hit"	"False Alarm"
Predicted Negative	"Miss"	"Correct Rejection"

Figure 15: Confusion Matrix

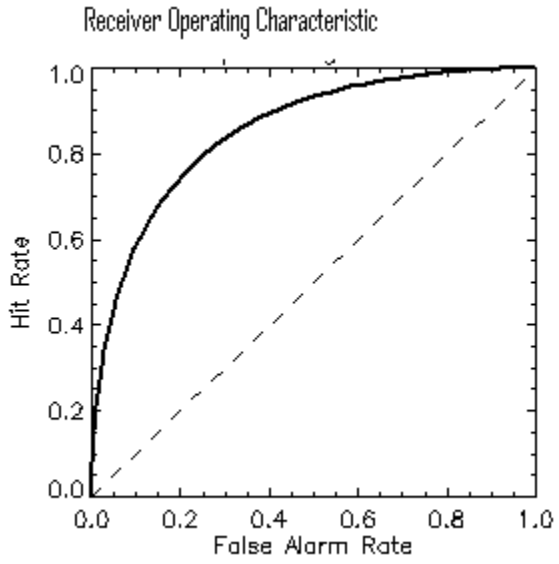

Figure 16: Receiver Operating Characteristic (ROC) Curve. (A Perfect Sensor Would Be A Vertical Line along the Y-Axis, Indicating A 100% Hit Rate and A 0% False Alarms Rate For All Discrimination Thresholds.)

CHAPTER 4 – THE COMPUTER VISION SYSTEM

The previous chapter focused on the biological vision system, and how object detection is conducted in humans. Unfortunately, most biological vision research is still in a rudimentary phase. Most computer vision engineers understand very little about the human biological vision system and have been performing experiments haphazardly. Therefore, many of the accomplishments in computer vision have come through painstaking trial and error; and finding a unified source of information remains a problem even with the widespread use of the internet. Nevertheless, there are some computer vision fundamentals that are common to every practitioner. The beginning of this chapter will cover some of these fundamentals in order for the reader to have a general knowledge of modern computer vision systems.

Computer vision is the science of machines that see. The goal of computer vision is for a machine to make decisions based on information from images. Several references use the terms machine vision and computer vision interchangeably, however there is a subtle difference. Machine vision usually refers to the context of industrial applications, where computer vision refers to the field in general.

The field of computer vision is a very broad one. This field combines the knowledge from various disciplines such as neurobiology, mathematics, physics, and artificial intelligence. Thankfully, one doesn't have to be an expert on all these topics to perform meaningful computer vision research. Because the computer vision field is so vast, most people simply specialize in one of the many subtopics of computer.

Computer vision has dozens of subtopics. Some prominent areas of computer vision include the following: 1.Motion 2.Scene Reconstruction 3.Image Restoration 4.Recognition. Several of these fields overlap each other. Because of this fact, most computer vision engineers will become experts at one subtopic, while having a mediocre knowledge about the other fields. The rest of this chapter will give an explanation on these four fields of computer vision with an emphasis on object recognition.

1. *MOTION*

Image formation for a camera is very similar to that of a human's. The main difference is that most cameras use a charge-couple device (CCD) instead of the rods and cones used by the human eye. The principle is essentially the same: tiny solid state cells sensitive to certain wavelengths of light convert them to electrical energy. This pixel coordinate system is usually organized in a raster orientation instead of a Cartesian system (**see Figure 17**). Almost every other principle remains the same as the same as the human eye.

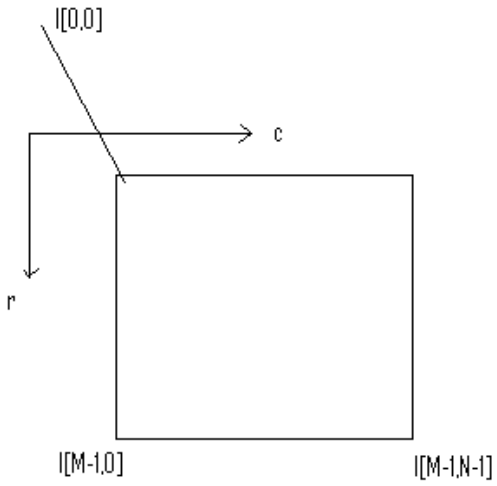

Figure 17: Raster Orientation

Motion describes the movement between an object of interest and the camera. The four basic cases of motion are described below from least to most complex.

1. Still camera, single moving object, constant background.
2. Still camera, several moving objects, constant background.
3. Moving camera, relatively constant scene.
4. Moving camera, several moving objects.

For the first case with a still camera, the movement of objects can be computed by simply subtracting two frames taken some small time apart. If the difference between pixels in two frames is above a threshold, we say a change occurred for those pixels. If only one object is moving in the field, that object can be identified using various simple algorithms, such as finding the center of mass of the moving object, or by using connected component analysis [18].

However, if the object is stationary, then subtracting two images won't give any pertinent information about the object being tracked. In this scenario, it is necessary to use some sort of point correspondence. Interesting *feature points* must be identified on the object of interest. These same feature points are then found in the next image and their motion is then calculated. This technique of tracking feature points is called *optical flow* (see **Figure 18)**. The most common types of feature points are corners or rough edges. The method for finding corners or edges will be covered later in this chapter under the section of Image Restoration.

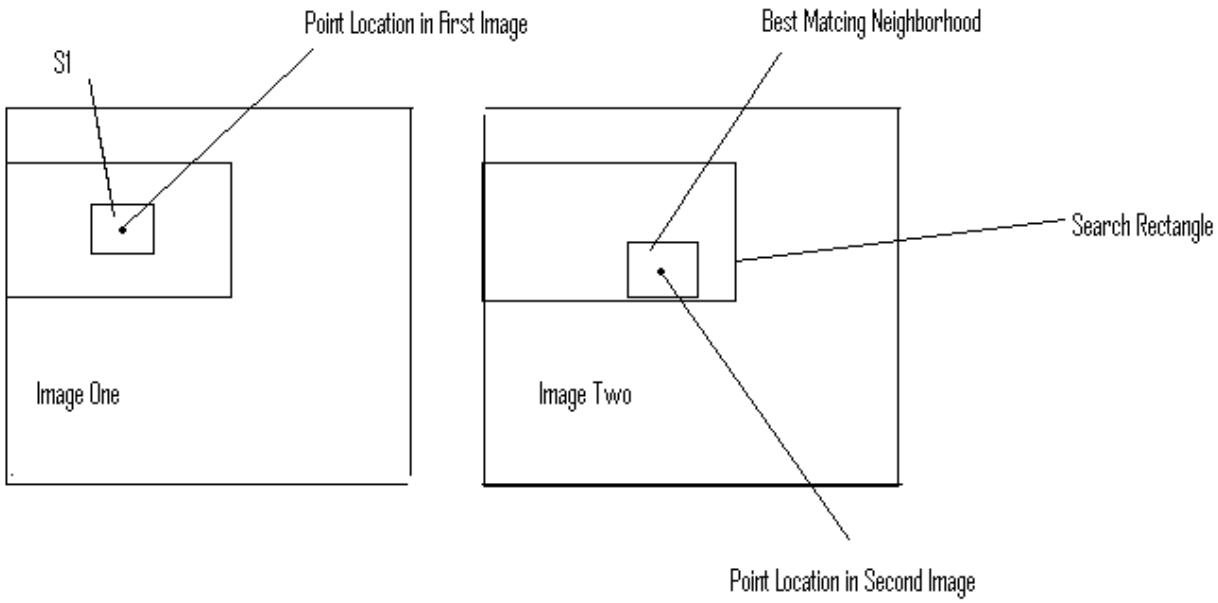

Figure 18: Point Correspondence for Optical Flow

Using a method such as corner or edge detection, a point of interest is found in the first image. A small region S_1 surrounding the point of interest is stored into memory. Now a second image is obtained. The stored region S_1 is then moved across a search rectangle is the second image. Wherever the stored region S_1 has the highest autocorrelation (least difference) in the search rectangle of the second image is the new location of the point of interest.

Computing the path of several moving objects becomes a more difficult task. Because several objects moving in the same image have the annoying tendency to cross paths, it is necessary to obtain a way to maintain some consistency of the tracked objects (see **Figure 19)**. In order to track multiple objects in the same image four assumptions are made. First, the location of any object changes smoothly over time. Second, the velocity of any object changes smoothly over time. Third,

any object can be at only one location at any given time. Fourth, two or more objects cannot occupy the same location at the same time.

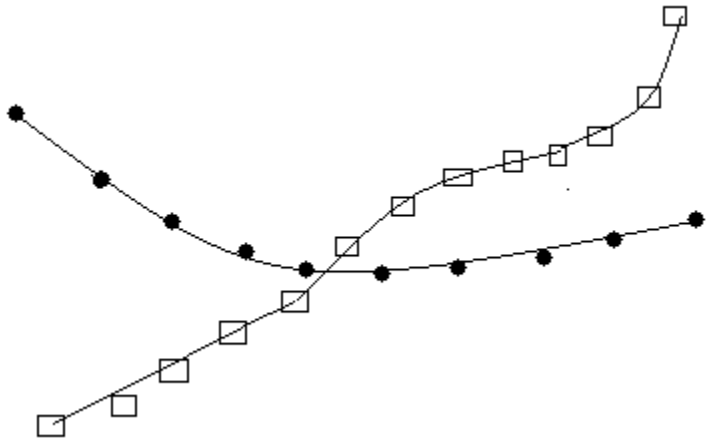

Figure 19: Finding Motion of Multiple Objects

Because three-dimensional space is projected onto a two-dimensional pixel plane, the fourth assumption may be violated if one object gets occluded by another. Regardless of this "minor" flaw, an algorithm has been developed by Sethi and Jain in 1987 to track multiple objects [19]. This algorithm uses the previous assumptions to compute the smoothest set of paths though points observed in a sequence of frames.

If an object i is observed at time instants t = 1,2,…n, then the sequence of image points T_i = ($p_{i,1}$, $p_{i,2}$,…$p_{i,t}$,…,$p_{i,n}$) is called the trajectory of i. The difference between two points is defined as **Equation 1**.

$$V_{i,t} = p_{i,t+1} - p_{i,t} \quad (1)$$

The smoothness value at point $p_{i,t}$ can be defined in terms of the difference of vectors reaching and leaving that point. The smoothness of direction is measured by their dot product. The smoothness of speed is measured by comparing the geometric mean of their magnitude to their average magnitude. The geometric mean is shown in **Equation 2**. Unlike an arithmetic mean where one adds a set of numbers and divides by the count, in a geometric mean one multiplies a set of n numbers and takes the n^{th} root of the result.

$$\left(\prod_{i=1}^{n} a_i\right)^{\frac{1}{n}} = \sqrt[n]{a_i \cdot a_2 \cdot \cdots \cdot a_n} \tag{2}$$

By using the definition of the geometric mean for two corresponding points, the smoothness can be defined as **Equation 3.**

$$S_{i,t} = w\left(\frac{V_{i,t-1} \circ V_{i,t}}{\left|V_{i,t-1}\right|\left|V_{i,t}\right|}\right) + (1-w)\left(\frac{2\sqrt{\left|V_{i,t-1}\right|\left|V_{i,t}\right|}}{\left|V_{i,t-1}\right|\left|V_{i,t}\right|}\right) \tag{3}$$

As one can see from **Equation 3**, the weight $0 \le w \le 1$ can be adjusted to emphasize smoothness of the direction or speed. The corresponding smoothness S will also lie between 0 and 1, with a higher value of S being better. The total smoothness for all points is given by **Equation 4.**

$$T_s = \sum_{i=1}^{m}\sum_{t=2}^{n-1} S_{i,t} \tag{4}$$

Points of interest in the first frame are labeled $i = 1,2,\ldots,m$. Given the next frame, the goal is to label points in the next image that maximizes the value of the total smoothness. There are several algorithms available that perform this procedure such as the Greedy-Exchange Algorithm [20].

The third case of motion occurs when the camera is moving, but the scene is relatively stationary. Optical flow methods have been developed for computing the flow of an entire image and not just at interesting points. These methods also have their corresponding assumptions. First, it is assumed that the object reflectivity does not change over the interval [t1,t2]. Second, the distances of the object from the camera or light sources do not vary significantly. Third, any small intensity neighborhood $N_{x,y}$ (e.g. any 3×3 or 5×5 set of pixels) at time t_1 is observed is some shifted position $N_{x+\Delta x, y+\Delta y}$ at time t_2. Unfortunately, these first two assumptions rarely hold in a real world setting. Unlike the scenario of tracking a single object with a stationary camera, moving the camera drastically changes the image. Nevertheless, an *image flow equation (***Equation 5**) has been developed to compute image flow vectors.

$$f(x+\Delta x, y+\Delta y, t+\Delta t) = f(x,y,t) + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y + \frac{\partial f}{\partial t}\Delta t \qquad (5)$$

The solution of this equation for Δx and Δy does not give a unique solution, but puts a linear constraint on it. Moreover, this differential equation must be solved for every pixel in the image making it computationally unfeasible for real world environments. Similarly, the fourth case of motion involves a moving camera and moving objects in the image. The analysis for this situation is very complex and error prone.

2. SCENE RECONSTRUCTION

The second broad topic in computer vision is scene reconstruction. The goal of scene reconstruction is to construct a three-dimensional model of a scene based on two or more *related* images. The reconstruction process consists of three main steps:

1. Three-Dimensional Data Acquisition
2. Registration
3. Surface Construction

In the *data acquisition* phase, range data is acquired from a set of views. In the *image registration* process, the range data is combined by transforming them all into a single 3D coordinate system. After the image registration phase is complete, the result is a three dimensional point cloud. A mesh is wrapped around this point cloud in the surface reconstruction phase.

Although the topic seems very novel, it is plagued with problems. First consider the data acquisition phase. It makes sense that a person should use the sensor which is appropriate for the job. A person wouldn't use sonar to check the temperature; nor would a person use a thermometer to spy on a conversation. Attempting to get range (distance) information from a monocular camera is similar to the above scenarios. Humans have an enormous perceptual set of objects stored into memory. Humans have prior knowledge of the size of millions of objects; computers do not. By identifying an automobile, a human has a rough approximation of the distance of that automobile based on its size relative to the viewer. Until computers have the storage and processing power comparative to that of a human's, trying to get range information from a monocular camera will be very inaccurate. If one needs range information, there are a myriad of sensors designed to give exactly that information. Second, the image registration process is usually never automated. Unless

the images are taken in a very controlled environment (i.e. placing infrared LEDs strategically throughout the image), a human must be present to aid the computer in transforming the information into a single three-dimensional coordinate system. Since the goal of computer vision is for the *machine* to make important decisions, having a human in the loop for anything but the most controlled scenarios essentially violates the definition. One would get much more accurate results by using a laser-range finder and building a mesh from the resulting point cloud.

3. IMAGE RESTORATION

Image restoration is the enhancement of images for either human consumption or further machine analysis. Image restoration is essentially image processing, since the input and output are both images. The formal definition of image enhancement is to improve the detectability of important image details. Image restoration is the process of restoring a degraded image to an ideal condition. These differences are very subtle, and the two terms will be used interchangeably for the rest of this paper.

The first type of image enhancement to be discussed is the contrast stretching operator. This operator essentially broadens the range of gray levels in an image (see **Figure 20 and Figure 21**). The most common type of contrast stretching operator is histogram equalization. Histogram equalization spreads the intensities of the image throughout the histogram [21].

Figure 20: Image before Histogram Equalization

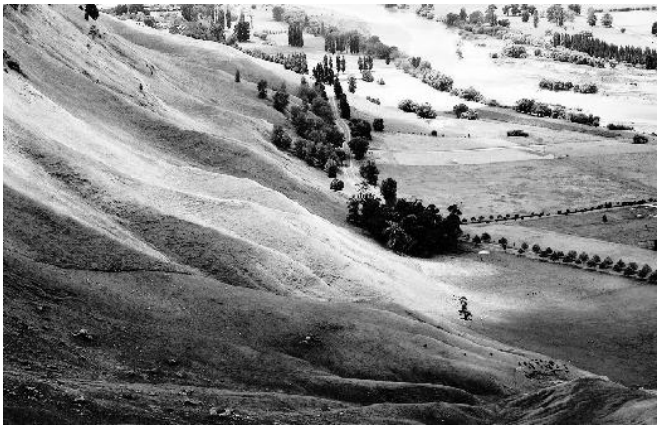

Figure 21: Image after Histogram Equalization

Histogram equalization is accomplished by the cumulative distribution function (cdf). Using the cdf, the original pixel value is mapped onto a new value. The cdf is shown in **Equation 6**. The mapping function is shown in **Equation 7**.

$$F(x) = \sum_{x_i < x} p(x_i) \tag{6}$$

$$map(v) = round\left[\frac{cdf(v) - cdf_{\min}}{M * N - cdf_{\min}} * (L-1)\right] \quad (7)$$

where,

M*N := The number of pixels in the image

L := Number of GrayScale Levels in the Image

As an example, consider the following mock 5×5 image with grayscale intensities shown in **Figure 22.** The goal is to equalize this image so that the values which are currently between the range of 17-27 are better distributed on the range 0-255.

20	19	25	21	23
22	24	24	22	25
21	24	23	23	19
23	24	22	24	18
27	22	21	25	17

Figure 22: Mock 5x5 Grayscale Image

The number of occurrences for each intensity value is now counted, as shown in **Figure 23**.

Value	Count	Value	Count	Value	Count
17	1	21	3	25	3
18	1	22	4	26	0
19	1	23	4	27	1
20	1	24	5		

Figure 23: Pixel Intensity Count

Now the cdf is computed. This is simply a sum of the counts that fall below a certain value. These values are shown in **Figure 24.**

Value	cdf	Value	cdf	Value	cdf
17	1	21	7	25	23
18	2	22	11	26	23
19	3	23	15	27	24
20	4	24	20		

Figure 24: Cumulative Distribution Function

Now that the cdf of the image is known, **Equation 7** is used to map the old pixel values into a more "spread" version. For example, to calculate the new value of a pixel valued "23", the following is performed.

$$map(23) = round\left(\frac{cdf(23) - cdf_{\min}}{(M \times N) - cdf_{\min}} \times (L-1)\right) \tag{8}$$

$$map(23) = round\left(\frac{15-1}{(5\times 5)-1} \times (256-1)\right) = round(148.75) = 149 \tag{9}$$

Therefore, each pixel with the value of "23" will be changed to a new value of "149". This process is done with every pixel in the image and the resulting image will have a much higher contrast than the original. However, this operation only produces significant changes if the original image has gray values that are close in intensity.

Image smoothing is another popular operation in image enhancement. If an image has some underlying structure that is to be enhanced and noise that is to be removed, smoothing operations can be used. There are three main types of smoothing operations. The first is the *box filter*. The box filter is the most basic of all the filters. It simply averages a 5×5 region around a pixel, and assigns the average value to the pixel (**Equation 10).** The image in **Figure 25** is full of salt-and-pepper noise.

Figure 25: Image Containing Salt and Pepper Noise

$$Out[r,c] = \left(\frac{\sum_{i=-2}^{2} \sum_{j=-2}^{2} In[r+i, c+j]}{25} \right) \quad (10)$$

After applying the box filter, the image in **Figure 26** is obtained.

Figure 26: Application of Box Filter

By looking at the resulting image, one can see that it did very little in getting rid of the noise. Even worse, it caused the image to become blurry. A *Gaussian filter* (**Equations 11 & 12)** can also be used to eliminate noise, but it has essentially the same shortcomings as the box filter. The Gaussian filter simply takes the values of the adjacent pixels and multiplies by a weighting factor g(x,y).

$$g(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-d^2}{2\sigma^2}} \tag{11}$$

Where,

$$d = \sqrt{(x - x_c)^2 + (y - y_c)^2} \tag{12}$$

The most commonly used filter for reducing "salt and pepper" noise is the *median* filter. Instead of averaging a 5×5 region around a pixel, the number directly in the center of the 25 intensity values will be substituted for the pixel. The result of the median filter is shown in **Figure 27**. As one can see, the "salt and pepper" noise is virtually eliminated from the original image.

Figure 27: Application of Medium Filter

Another major topic in image enhancement is edge detection. An edge is simply an image point of high contrast (high derivative). This point forms the border between different objects or abrupt changes in scenery. Most edge detection is done using neighborhood template or *masks*. Masks are also sometimes referred to as kernels. **Figure 28** shows how a mask is applied to a pixel.

-1	1	1
-1	2	1
-1	1	1

10	15	20	25	30
11	16	21	26	31
12	17	22	27	32
13	18	23	28	33
14	19	24	29	34

Figure 28: Application of Mask to a Pixel

For each pixel in the image, the 3×3 mask is convoluted to the pixel and its surrounding area. For example to apply the 3×3 mask to the pixel marked with intensity "16", the following calculation is performed: [(-1*10)+(-1*11)+(-1*12)+(1*15)+(2*16)+…]/sum(mask).

Dividing by the sum of the mask is sometimes left out of the computation. To see how a mask is used to compute the contrast of an image, a one-dimensional case is considered (**Figure 29)**.

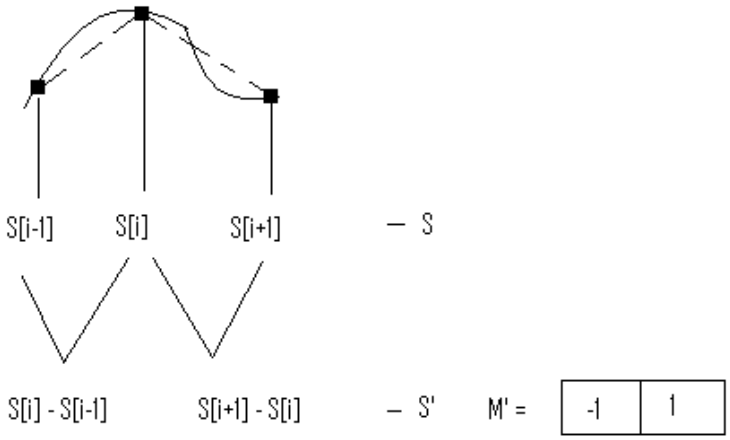

Figure 29: Computing Derivative of a Discrete 1-D Signal

Given that the signal S is a sequence of samples from some function f, the derivative is approximated as **Equation 13**. If the sample spacing is 1, then the mask M = [-1 1] can also approximate the derivative.

$$f'(x_i) \approx \left(f(x_i) - f(x_{i-1})\right)\left(x_i - x_{i-1}\right) \quad (13)$$

Similarly, the derivatives for 2D images can be computed in the same manner as for the 1D signal. However, there is one abnormality. Because the edge might cut across the pixel array at an angle (**Figure 30)**, it helps to average at least 3 estimates of the contrasts as shown in **Equations 14&15.**

$$\frac{\partial f}{\partial x} \approx \frac{1}{3}\left[\left(I[x+1,y] - I[x-1,y]\right)2 + \left(I[x+1,y-1] - I[x-1,y-1]\right)2 + \left(I[x+1,y+1] - I[x-1,y+1]\right)2\right]$$

$$\frac{\partial f}{\partial y} \approx \frac{1}{3}\left[\left(I[x,y+1] - I[x,y-1]\right)2 + \left(I[x-1,y+1] - I[x-1,y-1]\right)2 + \left(I[x+1,y+1] - I[x+1,y-1]\right)2\right]$$

(**Equation 14 & 15).**

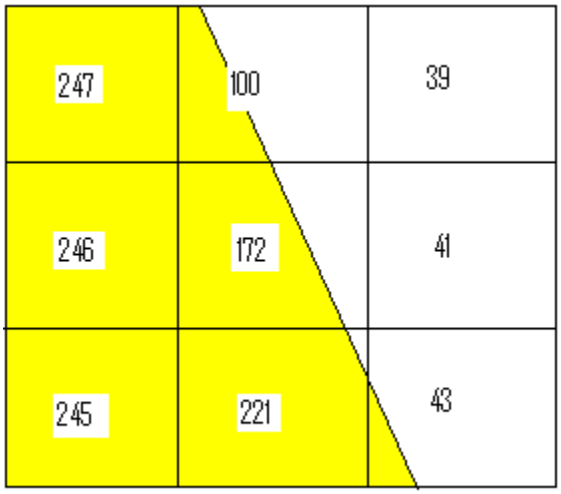

Figure 30: Edge at an Angle

Using **equations** 14 and 15, masks can be assembled to compute the derivatives of images. Two of the most popular masks are the Prewitt and Sobel masks. Both masks are shown below in **Figure 31**. The masks are virtually identical except that the Sobel filter emphasizes the center estimate.

Prewitt

Mx =

-1	0	1
-1	0	1
-1	0	1

My =

1	1	1
0	0	0
-1	-1	-1

Sobel

Mx =

-1	0	1
-2	0	2
-1	0	1

My=

1	2	1
0	0	0
-1	-2	-1

Figure 31: Prewitt and Sobel Masks

There are literally dozens of masks that can be used to modify images, such as the ripple mask (**Figure 32)** which is used for detecting corners in optical flow algorithms. For a more complete reference of masks used for image processing, one should read Digital Image Processing by Gonzalez and Woods.

$$W = \frac{1}{\sqrt{8}} * \begin{bmatrix} 0 & -1 & 1.41421 \\ 1 & 0 & -1 \\ -1.41421 & 1 & 0 \end{bmatrix}$$

Figure 32: Mask Used for Corner Detection

4. RECOGNITION

Pattern Recognition is a subtopic of machine learning. It is the act of taking in raw data and taking an action based on the "category" of the pattern. Pattern Recognition is a very broad field that encompasses making decision about numerous types of data. Some of the applications of Pattern Recognition concepts include stock market predictions, speech recognition or Signal Detection from devices such as radars. However, here the discussion of Pattern Recognition Concepts will be restricted to the subject of Computer Vision. In many problems, one needs to make decisions about whether an image contains a certain object. Sometimes the object is already given, and it is necessary to decide to which class the object belongs. Unlike most Expert (knowledge-based) systems which rely on the analytical skills of one or more experts in the field, the Pattern Recognition approach is normally a statistical one [22, 23].

Neural networks are also an approach to Pattern Recognition that is very popular. Neural networks are systems that try to mimic the neurons in the human brain. They consist of a set of neurons (simple computation elements) that are interwoven in a complex manner to exhibit complex global behavior [24], (see **Figure 33)**.

The main problem with neural networks is that they are very specific. They must be trained for every situation that will be encountered in a test. Given two outcomes, a probabilistic method will pick the outcome that has the highest chance of occurring. This more often will result in a correct decision. The same cannot be said about neural networks. If the neural network encounters a situation that it has not specifically trained to handle, the outcome will be unknown.

In computer vision using real images, there are simply too many variables to account for every scenario. Uncontrollable factors such as weather, camera vibration, and time of day can all affect the accuracy of a neural network system. Because of this fact, a neural network system used for computer vision must use vast amounts of training data. For example, training a neural network for face detection required around 10,000 images for good results [25, 26].

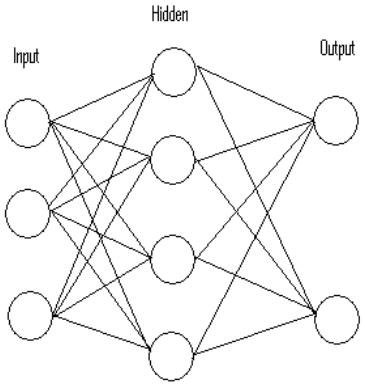

Figure 33: Simple Neural Network

The general steps in most Pattern Recognition systems are shown in **Figure 34**. Some pattern recognition systems might omit some of these steps, while others add steps of their own. Other pattern recognition systems may include feedback between one or all the steps. The importance of these steps will be illustrated using the example of the motorcycle in **Figure 35**. For this example, let's assume the goal is to find the motorcycle in the image and classify it as either a sport bike or a cruiser.

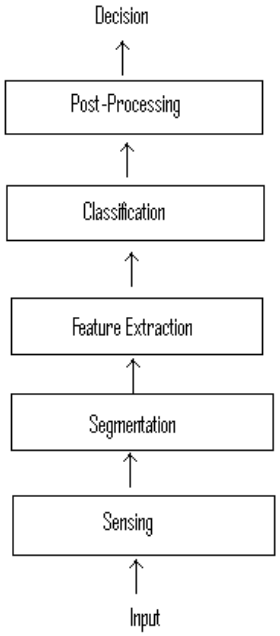

Figure 34: General Pattern Recognition Process

Figure 35: Extremely Fast Motorcycle with No Mirrors

SENSING

The first step in the Pattern Recognition Process is *sensing*. The sensor for most computer vision systems is a CCD or CMOS (Complementary Metal Oxide Semiconductor) camera. Both of these cameras take light energy and convert them to electrical energy. For this image, the sensing simply implies measuring the red, green, and blue wavelengths of light and combining them to form a color image.

SEGMENTATION

The second step in the Pattern Recognition process is *segmentation*. Segmentation is the partitioning of an image into a set of regions that cover it: segmentation is one of the most difficult problems in computer vision. Most pattern recognition methods assume the image is already segmented (pre-processed) and a decision needs to be made about the object of interest. However

this situation is rarely the case. For the test case image, before the motorcycle can be classified as either a sportsbike or a cruiser, it is necessary to find the motorcycle in the image.

Image segmentation usually requires expert knowledge of the image being analyzed. For example, if it is known that the object to classify is a blue motorcycle one can simply eliminate areas of the image that do not match this criteria, such as the green trees or the gray asphalt. There are numerous features (color, texture, etc) and algorithms that can be used for segmenting images. However, a preferred approach for finding objects in an image is to use a sliding search window. This approach will be discussed in the next chapter.

FEATURE EXTRACTION

The next step in the pattern recognition process is *feature extraction*. Feature extraction is one of the most important steps in pattern recognition. If the features used for classification are poor, then any corresponding classification scheme will also have poor results. Feature selection also requires expert knowledge of the object being classified; but unlike image segmentation, knowledge of the background is not needed. A good feature is one that is particular to one object and distinguishable to everything else. A good feature must also be *invariant*, meaning that it is still identifiable if the object undergoes some slight permutations. In the test case image, a good feature for separating a sports bike from a cruiser would be the solid blue color of the motorcycle (most cruisers don't come in solid blue.)

There are several features that can be used for detecting or classifying an object in an image. Some of these features include *color* and *texture.* These features will be discussed below. For this research, the approach will be limited to features that can be placed into histograms, as there are very useful techniques that can be used to analyze histograms.

Color is the form of electromagnetic radiation that falls roughly between the ranges of 400-700 nanometers. Most color in computer vision is done using the RGB (red, green, blue) color basis. Moreover, each value of color usually ranges in value from 0-255. Any arbitrary color can be made by combining different values of red, green, and blue. For example, red = (255,0,0), green = (0,255,0), black = (0,0,0), and white = (255,255,255). One can see the entire RGB color spectrum in **Figure 36**.

To compute a histogram of the image, one can simply divide the image into its red, green, and blue channels. To compute the histogram of a single channel image, one divides the x-axis into a series of bins. For a typical image, the range will be 0-255. Next each pixel in the image is considered. Given the intensity of the pixel, one simply adds a count to its corresponding bin. The results will look similar to **Figure 37.**

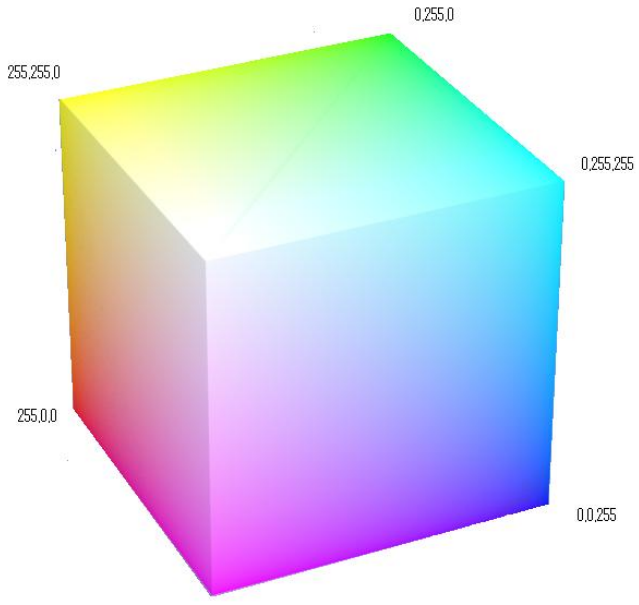

Figure 36: RGB Cube

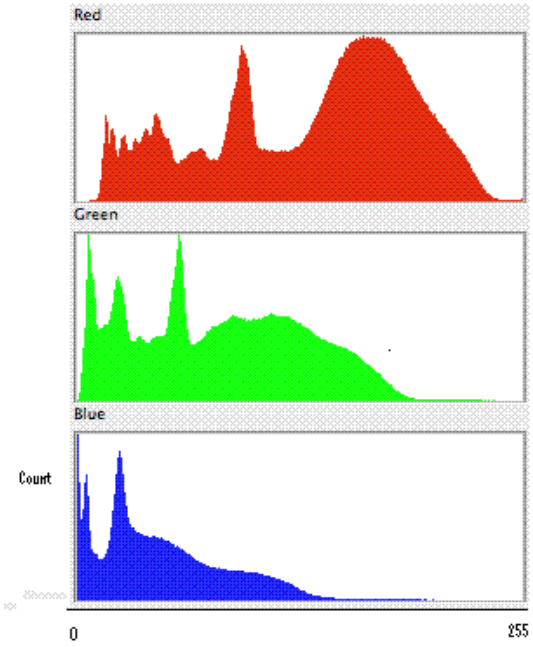

Figure 37: Histogram of Red, Green, and Blue Channels of an Arbitrary Image

Texture is another feature that can be used for object detection. Texture gives information about the spatial arrangements of the colors or intensities of an image. One measure of texture is the *edgeness per unit area* as defined by **Equation 16**.

$$F_{edgeness} = \frac{\left| \{ p \mid Mag(p) \geq T \} \right|}{N} \qquad (16)$$

This equation simply means that given an area (say a 5×5 kernel), count the number of pixels in that kernel that are above a certain threshold T. The results are then divided by the number of pixels N in the kernel. This process is repeated as the kernel is moved across the image. The results can then be

plotted in a histogram with the varying degrees of edgeness as the bin parameter. In **Figure 38**, the left image has an edgeness of 16 /25, while the image on the right has an edgeness of 21/25.

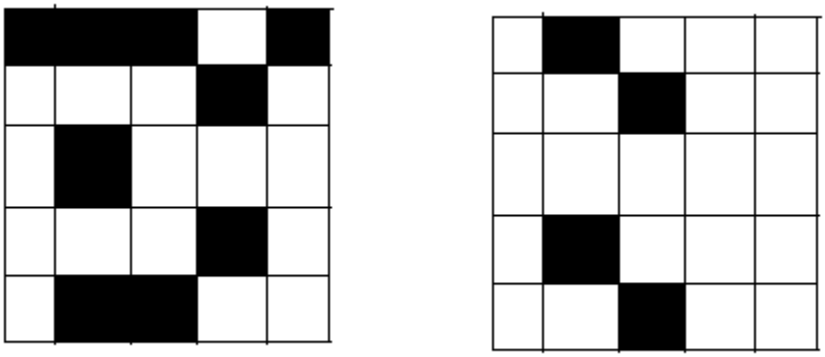

Figure 38: Measuring Edgeness of an Image

Another very simple, but very useful measure of texture is the *local binary partition*. For each pixel p in the image, the eight neighbors are examined to see if their intensity is greater than that of p. The results from the eight neighbors are used to construct an eight digit binary number $b_1b_2b_3b_4b_5b_6b_7b_8$, where, b_i=0 if the intensity of the i^{th} neighbor is less than or equal to that of p, and 1 otherwise. A histogram of these numbers is used to represent the texture of the image. While the local binary partition is a very simple texture measure, it is sensitive to changes in orientation of the image, such as rotation.

CLASSIFICATION

The fourth step in the pattern recognition process is *classification*. The job of the classifier is to use the features to assign the object to a category. There are literally dozens of classification

methods used to determine an object's class membership. A good reference for these methods is Pattern Classification by Duda, Hart, and Stork [27].

Most traditional classification techniques involve building a classifier using training data from each class. The user then takes a feature (e.g. color) that distinguishes the classes from each other, and builds a normalized histogram of the training objects (see **Figure 39)**. Once this histogram is constructed, the user can use any number of methods to determine the class membership of any new test images.

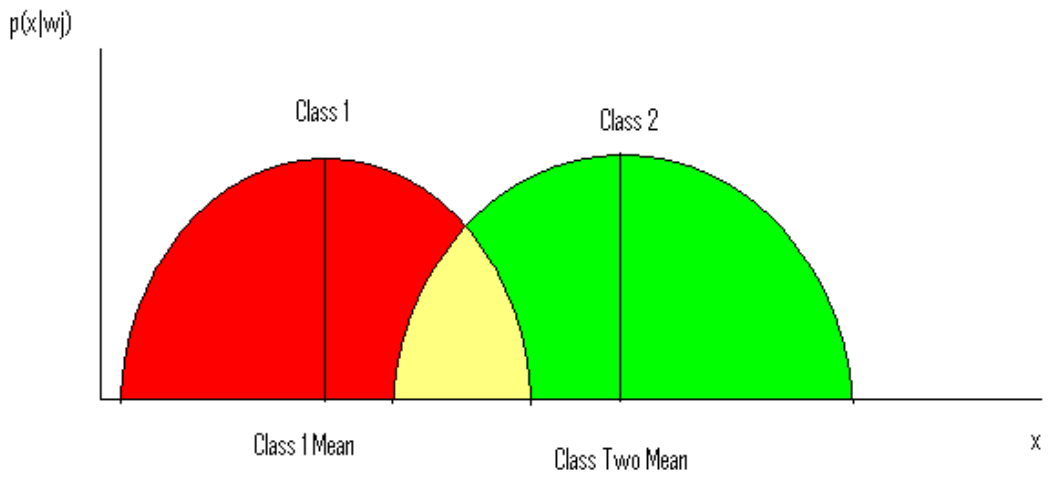

Figure 39: Normalized Histogram (Probability Density Function) of Two Classes

Some classification methods are as simple as the nearest mean. The test image is assigned to the class whose feature mean is closest. Other simple methods include the nearest neighbor analysis. This means that the test image is assigned to the class which has a data point closet to the test image (see **Figure 40**).

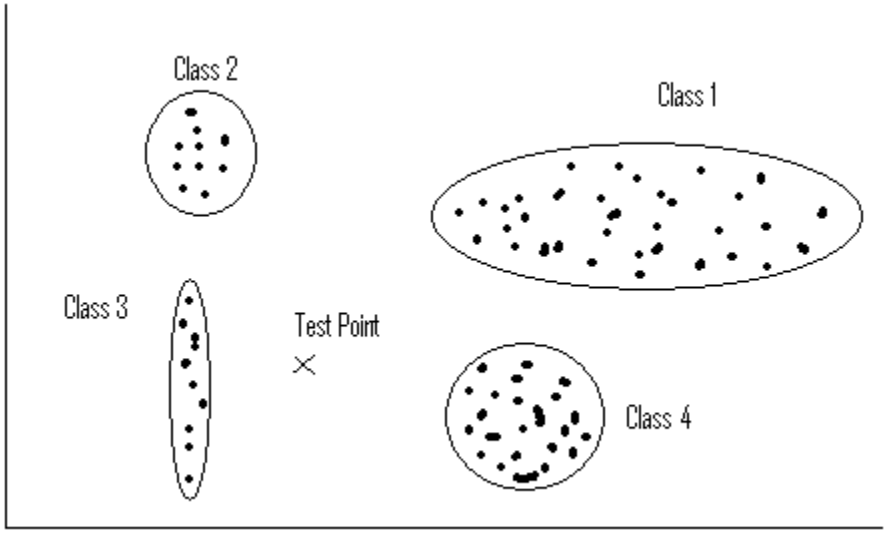

Figure 40: Nearest Neighborhood Analysis

More general methods include Bayes formula (**Equation 17&18**). This method allows the experimenter to scale his/her results by introducing prior knowledge. In the motorcycle example, if the experimenter knows he/she is more likely to see sportbikes than cruisers, he/she can scale the classification method appropriately.

$$p(w_j \mid x) = \frac{p(x \mid w_j) * P(w_j)}{\sum_{j=1}^{n} p(x \mid w_j) P(w_j)} \tag{17}$$

$$posterior = \frac{likelyhood * prior}{evidence} \tag{18}$$

The *likelihood* is exactly the same as a normalized histogram. The *prior* can be based on expert knowledge. The *evidence* is simply a scaling factor that ensures the posterior's value will lie

between 0 and 1. Given a test image, one can now calculate the posterior probability that it belongs to a certain class. The class that gives the highest posterior probability is chosen.

As stated before, there are several methods used for classification. Some methods assume the likelihood is Gaussian. Some methods allow us to include risk factors. Other methods such as the maximum-likelihood algorithms allows one to determine the parameters of the likelihood if they are unknown. Once again, the text by Duda, Hart, and Stork provides a good reference for these techniques.

POST PROCESSING

Post processing simply means that another step can be included before a final decision is made. In the test case example with the motorcycles, if the probabilities that it was a sportbike or cruiser were almost identical, it could be decided that it is necessary to take another action. Post processing is important in applications where a wrong decision could be disastrous. If one has an automated gun turret that is supposed to shoot anyone who's armed but let everyone else pass, a wrong decision would be disastrous. Therefore, one can set some threshold in post processing where the gun won't fire unless the probability that the person is armed is extremely high. Even better, the post processer could be a human who has the final say on whether to shoot.

MORE ON FEATURE SELECTION

As stated earlier, a pattern recognition system's "weakest link" is the features. If a human's choice of features is poor, no pattern recognition system will be able to overcome this hurdle. Feature selection is the most important part of designing any pattern recognition system, but yet it is the least talked about in most textbooks. On the other hand, there are hundreds of research papers

that address the topic of feature selection. This following paragraphs will address some of the most applicable topics.

In 1996, Richeldi and Lanzi introduced a program call ADHOC, which aids in the feature selection process [28]. This tool eliminated redundant and irrelevant features that could be detrimental to system performance. This program used tools such as *principle component analysis* to reduce a high dimensional data set into a smaller one. Principle component analysis reduces the dimensionality of a data set by retaining characteristics of the data set that contribute most to the its variance by keeping low-order principle components and ignoring higher-order ones. However, principle component analysis simply organizes the data in terms of highest variance. Principle component analysis tells the user which features are least correlated, so that the user can reduce the redundancy in his/her computation. The user must still determine which features are best for tracking the desired object. This simple transformation of features is generally called *feature extraction* rather than feature selection.

Also in 1996, Zongker and Jain performed a comparison of feature selection algorithms [29]. Some of these algorithms include *Sequential Forward Selection* and *Sequential Backward Selection*. Sequential forward selection starts with an empty feature subset. For each iteration, exactly one feature is added to the feature subset. To determine which feature to add, the algorithm adds one feature that is not already in the subset and tests the accuracy of a classifier built on the tentative feature subset. Sequential backwards deletion is similar to forward selection, but it starts with all the features and eliminates the features that result in a classifier with the least error. One of the best results from this research shows that each algorithm has poor results when the training data is small (less than 20 samples.)

In 1997, Lanzi introduced a way of using Genetic Algorithms to reduce the number of redundant features [30]. A genetic algorithm is search technique that finds solutions to optimization techniques. His research shows that this new genetic algorithm is at least 10 times faster than any previous genetic algorithm or principle component analysis.

In 2005 Liu, Dougherty and others once again showed that most feature selection algorithms performed poorly when the training data is small and the original dimensionality of the feature set is high [31]. Most research papers tend to agree that traditional feature selection algorithms based on supervised or unsupervised learning give poor results when given a high initial feature set and a small set of training data.

There are currently new methods being developed for feature selection using *information theory*. Information theory is a branch of mathematics and engineering that focuses on the quantification of information. One of the key concepts of information theory is the concept of *entropy*. Entropy quantifies the uncertainty of a random variable (**Equation 19 & 20)**. **Equation 19** shows how to calculate the entropy for a discrete probability density function, while **Equation 20** shows the entropy for the continuous case.

$$H = -\sum_{i=1}^{n} P_i \log P_i \quad \textbf{(19)}$$

$$H = \int_{-\infty}^{\infty} p(x) \ln p(x)\, dx \quad \textbf{(20)}$$

The *relative entropy* between two probability density functions (**Equation 21**) can also be used as a measure of reducing redundant features [32]. This measure of entropy between two probability

density functions is termed the KullBack-Leibler Divergence. This divergence will be further discussed in the next chapter.

$$D_{kl} = \sum_{i=1}^{n} q_i \log\left(\frac{q_i}{p_i}\right) \quad \textbf{(21)}$$

In summary, this chapter has shown that computer vision is a very broad topic. Several research topics have been presented such as image restoration and object recognition. The fundamentals of pattern recognition as it applies to computer vision have also been discussed. Feature selection is the most important part of any pattern recognition system and various research papers on this topic have been presented. These feature selection algorithms include traditional methods which fail for small sample sizes. A more novel approach of feature selection using information theory has also been presented.

In the next chapter, a new method for feature selection is proposed which can be used for very small sample sizes. This method will work regardless of the sample size or the number of initial features available for selection. Moreover, future users will be able to expand this feature set with minimal effort, and the feature selection method will be unaffected.

CHAPTER 5 – METHODOLOGY

This chapter proposes a method for feature selection that can be used on objects with relatively stationary statistics. This method for feature selection can also be implemented with small sample sizes and a high dimensional feature set. By incorporating a simple object-oriented approach, easy expansion of the feature set can be achieved. Moreover, this method used for feature selection doesn't require any traditional "training", which causes problems for so many feature selection algorithms.

The goal of this research is to vastly simplify the feature selection process. A set of pre-defined features are organized into a database. These features can be simple such as color, texture, shape, or complicated like Haar wavelets and features derived from principle component analysis. The user simply selects the object of interest in a minimum of two consecutive images; the method then informs the user of which features will be best for recognizing that object in future images.

The following is an example of feature selection using color. Assume the user wishes to track the Mustang shown in **Figure 41**.

Figure 41: Custom Mustang

The image of the Mustang is split into its corresponding Red, Green, and Blue channels as shown in **Figure 42**.

Figure 42: Red, Green, and Blue Channels of Mustang Image

A histogram of the red, green, and blue images is then constructed as shown in **Figure 43 -45.**

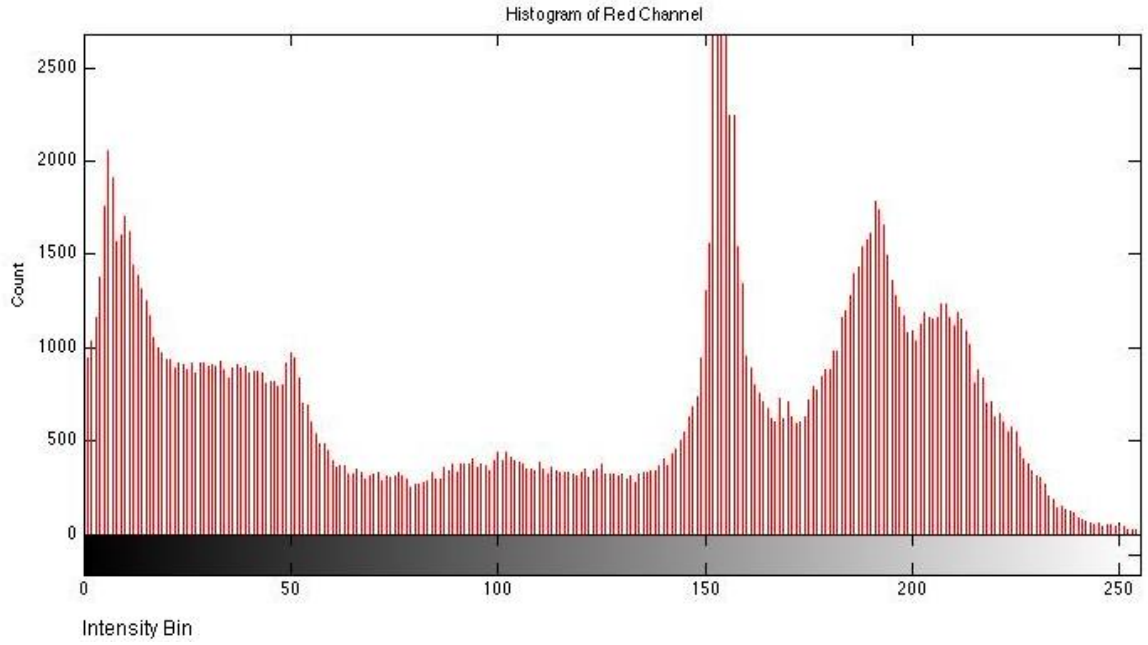

Figure 43: Histogram of Red Channel of Mustang

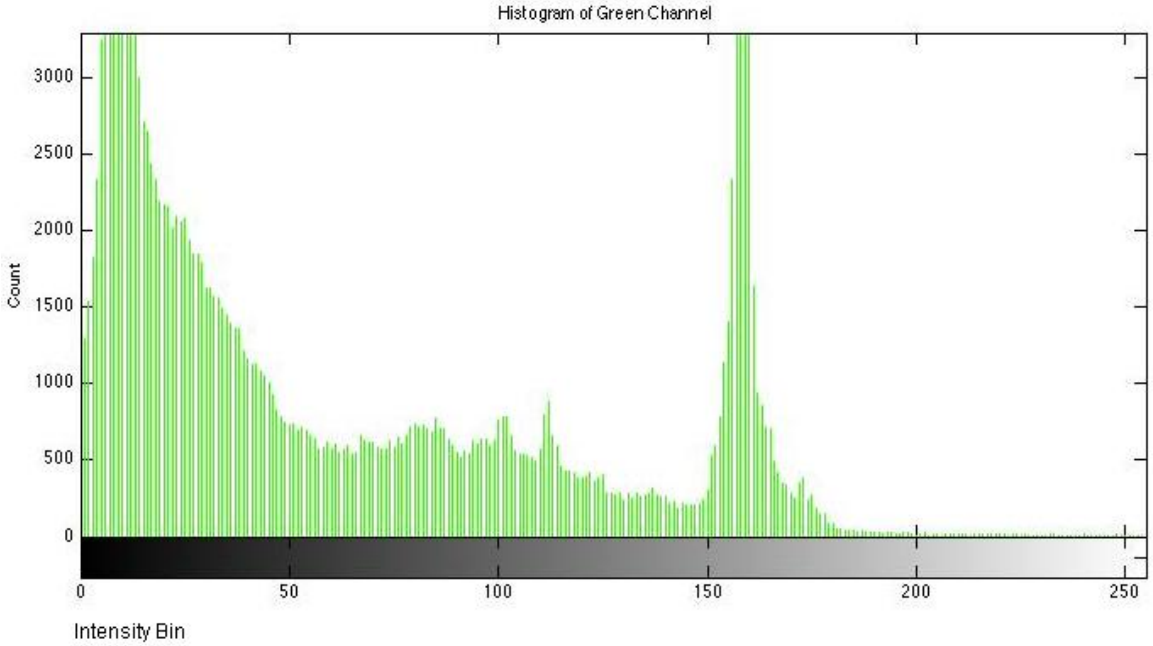

Figure 44: Histogram of Green Channel of Mustang

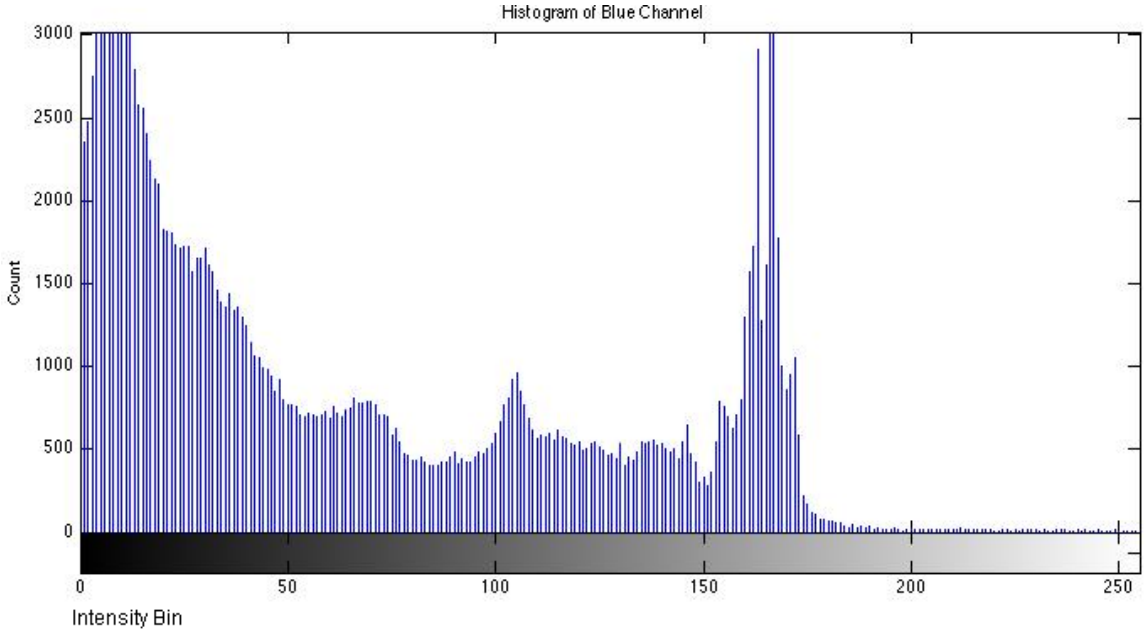

Figure 45: Histogram of Blue Channel of Mustang

The user then selects the object in another image (**Figure 46**).

Figure 46: Second Image of Mustang

Similarly, histograms are built using the red, green, and blue color channels of the second selected image (**Figure 47-49**).

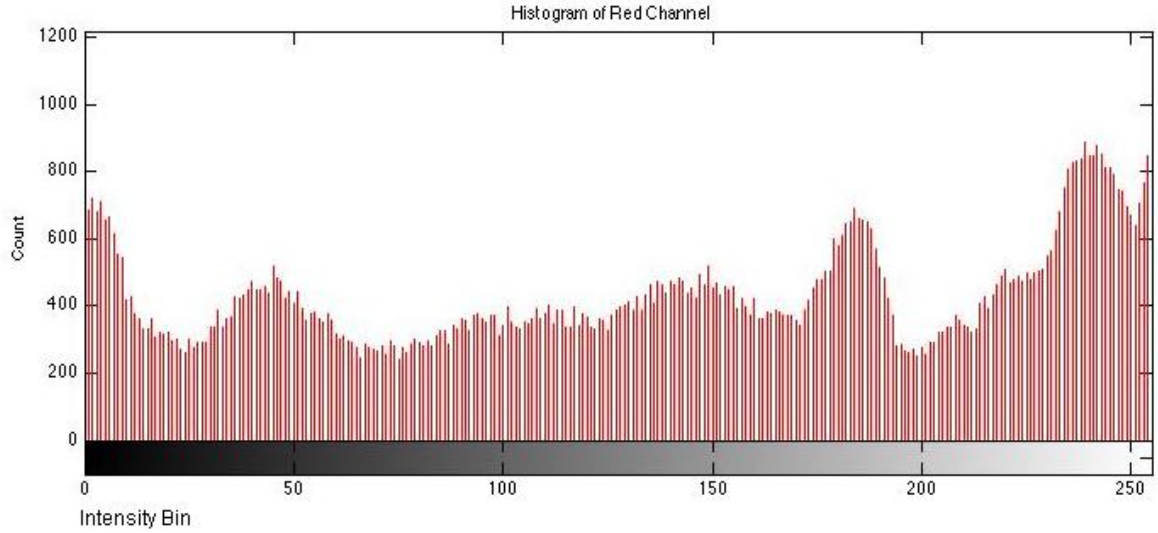

Figure 47: Histogram of Red Channel for Second Mustang

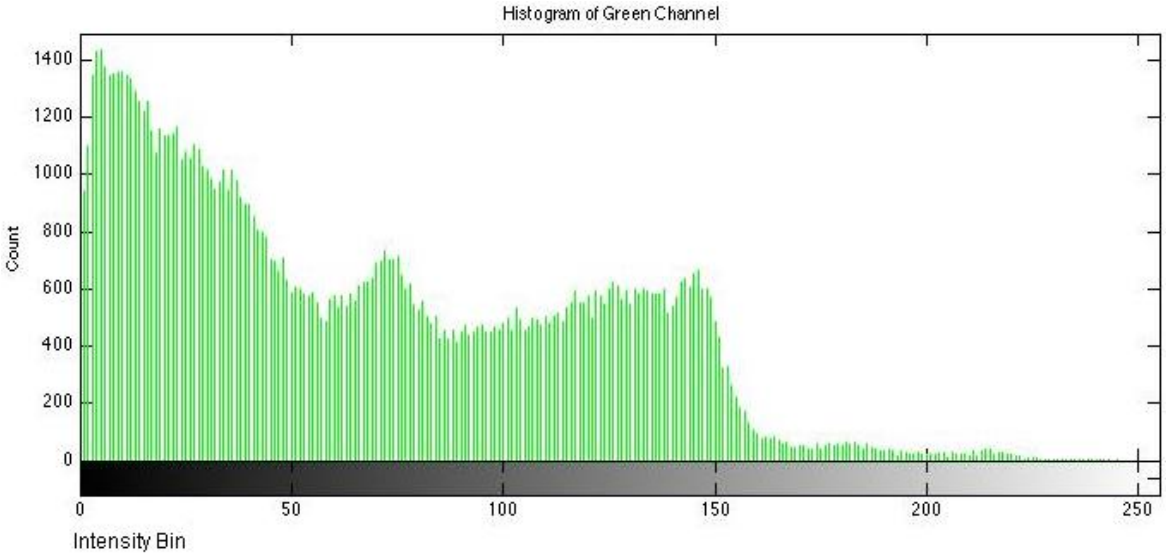

Figure 48: Histogram of Green Channel for Second Mustang

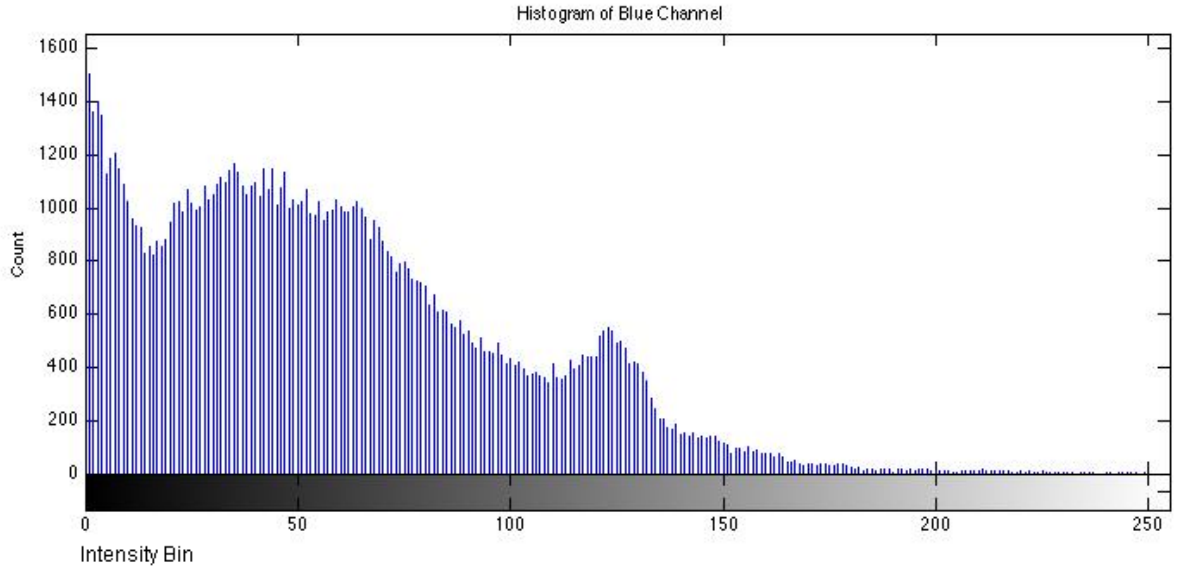

Figure 49: Histogram of Blue Channel for Second Mustang

Now that histograms of both images have been assembled, a way to compare these histograms must be developed. There are many methods that exist to compare the similarities of two histograms. Two methods that are used extensively for comparing Gaussian Histograms are the Euclidean distance and the Dunn Index (**Equations 22 & 23**). The Euclidean distance is simply the distance between the means of the histograms. The Dunn index takes into account the variance (spread) of the data. Unfortunately, most histograms taken from real images rarely have a Gaussian distribution. Therefore, these two methods of comparison will give poor results for most real applications.

$$d = \left[(\mu_2 - \mu_1)^2\right]^{\frac{1}{2}} \tag{22}$$

$$d = \frac{|\mu_2 - \mu_1|}{\sqrt{\sigma_1^2}\sqrt{\sigma_2^2}} \tag{23}$$

As stated in the previous chapter, recent approaches to feature selection use information theory. More specifically, these approaches rank the similarity between two histograms by using relative entropy (**Equation 21**). At first impression this relative entropy (Kullback-Leibler) divergence seems to be an excellent way to compare the similarities between two histograms. However it will later be shown in the *Feature Development* section why this distance fails for the scenarios considered here. Another excellent method for comparing the similarities of two histograms is the L_1 Norm, also known as the city block metric (**Equation 24**).

$$L_1 = \sum_{i=1}^{n} |O_i - E_i| \tag{24}$$

After the distance between histograms has been measured, the program checks to see if this distance is below some threshold. If the distance between histograms is below a certain threshold, the program selects the feature as "good." **Figure 50** shows a list of preliminary features that will be implemented for this research.

RGB ColorSpace	Image Weight	GrayScale Modal Analysis
GrayScale Intensity	Image Moment of Inertia	Texture Modal Analysis
HSV ColorSpace	RGB Modal Analysis	Number of Circles
Texture	HSV Modal Analysis	Number of Rectangles

Figure 50: Preliminary Feature List

As one can probably see, the method of measuring the distance between histograms will not be completely applicable to every feature in **Figure 50**. For example, storing the number of circles into a histogram would not yield any relevant information. Any variations in computing the distance measures will be described in the next chapter titled *Feature Development*.

Once the good features have been selected, the next step is to actually find the object in future images. For this approach, a growing, search window will be used as shown in **Figure 51**. It was shown in the previous chapter *The Human Vision System* that human sight is one of holistic nature, therefore making traditional pattern classification techniques non usuable. By using this growing search window, the holistic nature of the human vision system can be mimicked. Everything contained in the search window will be considered a possible object of interest. The properties of the region inside the rectangle will then be compared to previous features. This window will slide over the entire image. If the properties inside the search window are similar to the properties of the good features, the search window will return a "hit" for the object of interest. If the distance between the region inside the rectangle and the feature classes is too large, the region inside the rectangle will be rejected. As the search window is moved across the image, it will most likely report several hits for the same object as shown in **Figure 52**. To eliminate this redundancy, if a hit from a smaller search window lies inside a hit from a larger search window, the hit from the smaller search window is deleted (**Figure 53**).

Figure 51: Small Sliding Search Window

Figure 52: Example of Multiple Hits on the Same Object

Figure 53: Elimination of Multiple Hits

TESTING

As stated at the beginning of this chapter, this method will be useful for tracking objects that possess relatively stationary statistics. This statement implies that the object being tracked has statistics that will not vary in regard to each feature class. As an example, it is assumed that the object's texture and shape will remain relatively constant throughout the duration of any testing. In regards to texture and shape, solid objects of constant properties such as automobiles or textbooks will be great for testing. Amorphous objects such as a handful of rocks or water will not fair very well using this method.

Another assumption when using this method is that the environmental variables (mainly lighting) remain relatively constant. Although the user cannot completely control the environment, he or she should try to minimize any drastic changes that could negatively impact the distance measures for each feature class. In most practical situations, putting forth a modest effort to ensure constant lighting is more than enough to obtain reasonable results.

The accuracy of this method will be documented using a confusion matrix as described in (**Figure 15**). One of the assumptions in using this method is that the objects being tracked will have relatively fixed statistics. Because of this assumption, solid objects such as automobiles will be used in the outdoor testing scenarios. The other assumption used in this method is the environmental variables remaining relatively constant. Therefore, testing will be conducted in conditions that provide a constant and adequate lighting source. The optimal condition for outdoor testing will be midday under clear skies.

Lighting is usually not a problem for indoor testing scenarios as the source typically remains constant and can be controlled by the user. Once again, the object being tracked must possess relatively constant properties. Objects used for indoor tracking will be left to the discretion of the user.

The overall goal of this research is to drastically reduce the feature selection process. This method will be compared to two well known methods for object detection. These methods consist of artificial neural networks and template matching. As stated before, a neural network classifier provides laudable results when given adequate training data. However, gathering "adequate training data" is usually a very time consuming task as most neural networks require thousands of training images for appreciable results.

The second well known method that will be used for comparison is template matching. Template matching is a method often used in industrial processes where machines are used to inspect products under extremely controlled conditions. Examples of template matching include quality control applications where machines examine circuit boards for defects, or examine screws for thread defects. Template matching requires very little training. Unfortunately, the results of template matching are not useful unless the environment is completely controlled.

This new method is designed to give the user the robustness of a neural network classifier with the user-friendliness of a template matching algorithm. All three methods will be tested under the conditions described above. The results of all three methods will then be documented in the previously mentioned confusion matrix. The next chapter will show the development of the individual feature classes shown in **Figure 50.** After the creation of all feature classes are explained, the process of assembling the features classes into a fully working algorithm will be described. Finally the working program will be demonstrated in its entirety.

CHAPTER 6 - FEATURE DEVELOPMENT

This chapter details the steps in creating the program. The twelve feature classes to be used in the final program will each be given special attention. Any variations from the *Methodology* section will also be described in detail. After the creation of each feature class is demonstrated, the next chapter will give a brief overview of how the feature classes will be assembled into the final program.

1. RGB CLASS

The first feature class to be discussed is the Red, Green, and Blue (RGB) color space. As shown in **Figure 37**, a color image can be divided into its corresponding RGB color space. Once the image is separated, a histogram of each color space can be created. To determine whether or not the color space is useful for tracking, the L_1 and D_{KL} distances between two images of the same object are calculated. If the distances are below a certain threshold, then this color space will be selected as a candidate to find the object in future images.

The first step is determining the threshold to classify this color space as a usable feature. The distance between two objects is influenced by numerous factors including the following: Histogram Bin Size, Variations in Lighting, Shadows, and Object Transformations/Rotations. However, one of the assumptions in using this algorithm is that the lighting will be kept relatively constant. Because of this assumption, changes in lighting and shadows will be kept to a minimum. Rotations of the object of interest will not play a major factor in the distance calculations since all RGB distance measures are calculated from a histogram, which itself is a simple pixel value count.

The major factor affecting the distances between two objects is the histogram bin size. An appropriate bin size is one which minimizes the variations when tracking the same object, but will provide good separation between different objects. **Figure 54** and **Figure 55** show two very similar objects that could possibly be used for tracking.

Figure 54- First Image of Object to Track

Figure 55 - Second Image of Object to Track

Figure 56 shows how the L_1 and D_{KL} distance vary as the number of bins are increased for the objects in **Figures 54 and 55**. As expected, the distances increase as the number of bins increase. The figure shows the distance comparisons of the Red Channel as a function of bin size.

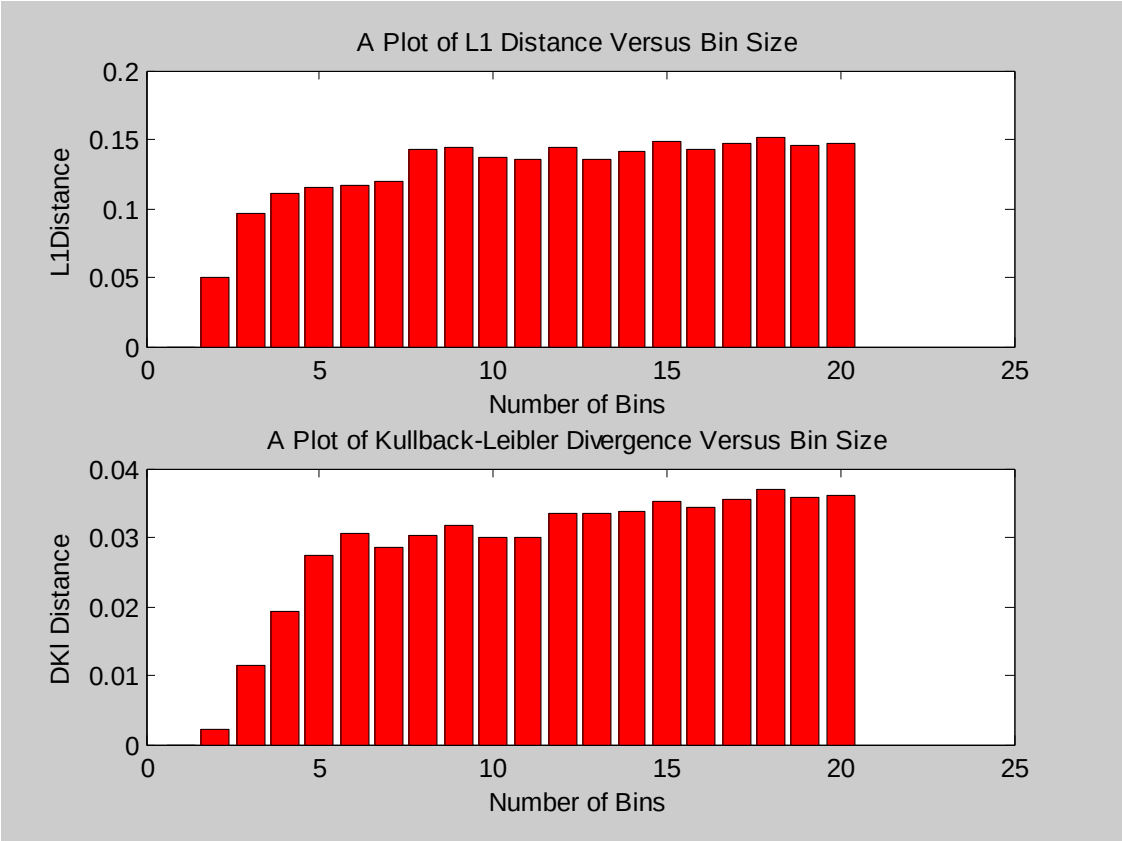

Figure 56 - Plot of Distance Measures versus Bin Size

As one can see, a plot of bin size versus either distance measure doesn't yield any helpful information. There is no "magic number" of bins that minimizes the distance between similar objects. To find a reasonable number of bins for tracking, a different approach is needed. Instead of focusing on how bin sizes affect the distance measures, the focus instead needs to be on the variations of a single object. By determining the color variations under normal circumstances, the bin size can be calculated in return. To accomplish this task, multiple images of a solid color will be taken in relatively constant lighting conditions. The only variations will be of orientation and distance. Although the lighting on the object will be indirectly affected by the object's transformations, direct manipulation of the light source will not be part of the testing.

To accomplish this testing for the RGB color space, a simple C++ GUI program was created. As shown in **Figure 57,** this program calculates the average red, green and blue intensity values for everything contained inside the green box.

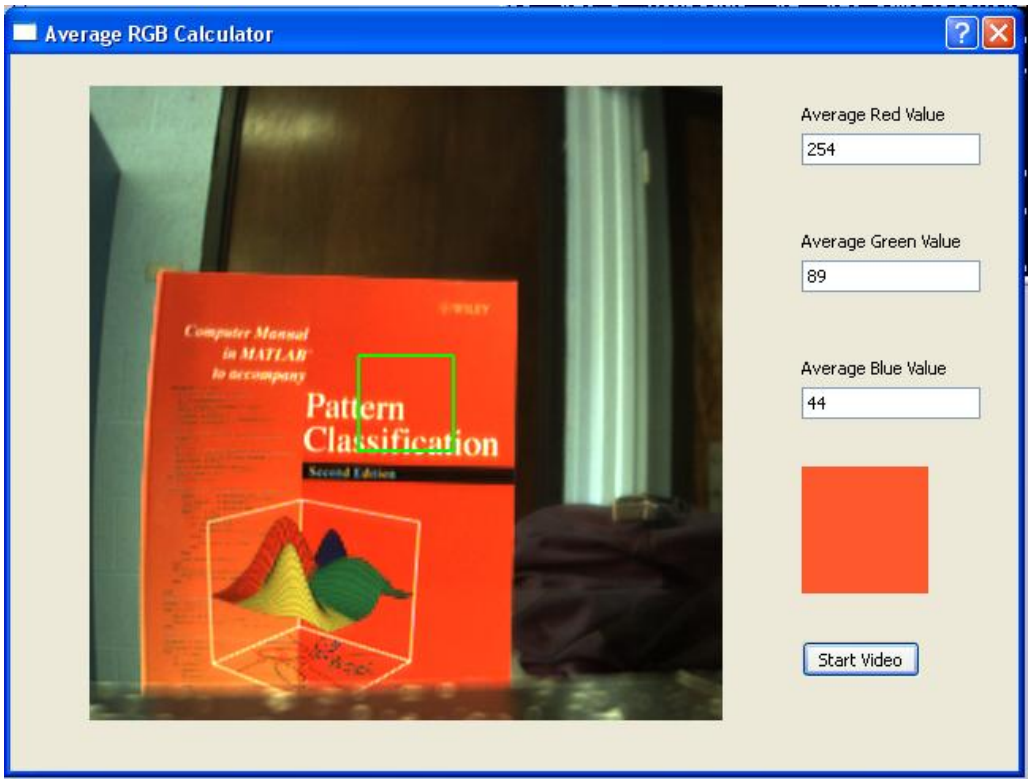

Figure 57 - Program Used To Calculate Average RGB Values

This tool will be used on several objects to determine the variations of color caused by the transformations of that object in space. The objects used for testing will be homogeneous in color and have constant properties as stated in the above testing scenarios. Also, this bin size analysis will be performed indoors as to minimize any changes in lighting.

Throughout the testing, the green channel almost always had the greatest variance of all three channels (Red, Green, and Blue). This result was expected as the green channel has the greatest

effect on an object's luminance as shown by **equation 25** (The National Television System Committee (NTSC) conversion of a color image into a grayscale). Because the only changes on the object were indirect changes in lighting, it makes sense that the green channel would have the biggest influence. **Figure 58** shows some of the sample variations that were encountered in the testing scenarios. The numbers in the figures are the minimum and maximum intensity values for each channel (Red, Green, Blue) while tracking an object of solid color. These values were obtained using the program in **Figure 57**.

$$GrayScale = 0.3 \times \mathrm{Re}\,d + 0.59 \times Green + 0.11 \times Blue \quad \textbf{(25)}$$

	Dark Green Object	
	Min	Max
Red	40	60
Green	70	107
Blue	44	66

	Gray Object	
	Min	Max
Red	160	179
Green	150	180
Blue	126	136

	Red Object	
	Min	Max
Red	250	255
Green	60	90
Blue	13	24

	Yellow Object	
	Min	Max
Red	250	255
Green	250	255
Blue	80	107

	White Object	
	Min	Max
Red	250	255
Green	250	255
Blue	250	255

	Dark Blue Object	
	Min	Max
Red	24	30
Green	40	60
Blue	40	60

Figure 58 - Sample Variations in RGB Values

As shown in **Figure 58**, the variations caused solely by transformations of the object were anywhere between 5-37 intensity values. From this observation, a bin size of 40 intensity values should encompass the changes caused by most spatial transformations of a desired object. This bin size of 40 intensity values will now be used in determining whether or not the RGB color space is a good feature to use for tracking.

Using the bin size of 40 intensity values, the next step in testing is determining the thresholds for the L_1 and D_{KL} measures in order to test whether a feature is reliable for tracking. The method used to determine the thresholds for the distance measures is the same as determining the bin size. Several random objects were selected and their orientation was shifted in space. Once again, the lighting in the testing was kept constant. The variations in the L_1 and D_{KL} were then calculated. A plot of the L_1 distances is shown in **Figure 59.**

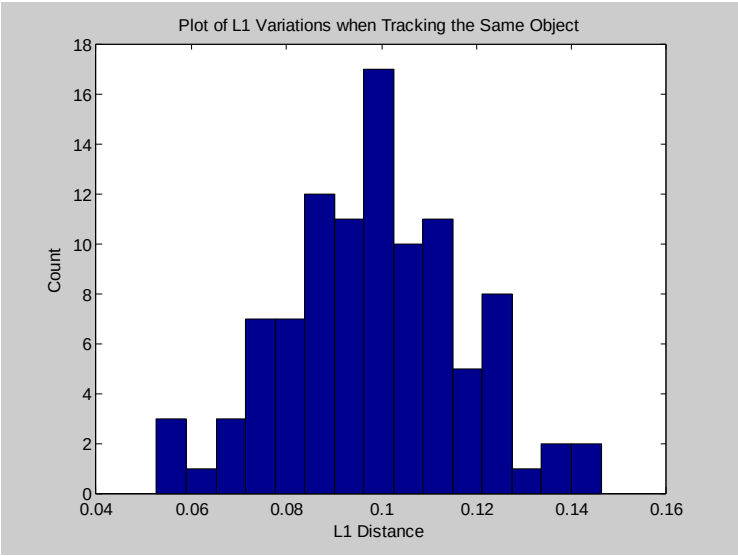

Figure 59 - Variations in L1 Distance while Tracking the Same Object

As shown in **Figure 59**, the mean L_1 distance (a normalized value between 0 and 1 for calculating the distance between histograms) when tracking the same object was around 0.1 units. The maximum L_1 distance encountered when repeatedly tracking the same object was slightly above 0.15 units. Based on these results, a very generous L_1 threshold of 0.25 units was chosen. This threshold should be more than enough to compensate for any variations caused by simple geometric transformation of the object.

NOTE: The selected bin size and L_1 distance is a tunable parameter that is completely left to the discretion of the user. Prior knowledge of the objects destined to be tracked would help in selecting these parameters. Since this application is designed to be very general in the objects being tracked, the selected bin size and L_1 distance will be slightly larger than anything encountered in testing.

Several problems were encountered when calculating a threshold for the D_{KL} distance. The first major problem is that D_{KL} is nonlinear. This nonlinearity makes determining a threshold for a general set of objects very impractical. Second, because of the nature of the logarithmic function, if a bin contains zero counts, the D_{KL} distance will explode to infinity. This problem with the D_{KL} distance makes it unreliable and not very useful for classification. Because of these problems, the D_{KL} as defined again in Equation 26 will not used in the application development.

$$D_{kl} = \sum_{i=1}^{n} q_i \log\left(\frac{q_i}{p_i}\right) \qquad \textbf{(26)}$$

In summary, the results of testing for the RGB class show that a bin size of 40 accounts for most variations in solid colors. The testing also showed that an L_1 threshold of 0.25 should account for most geometric transformations of solid objects. These values for the RGB class will be used in the fully working application.

2. GRAYSCALE CLASS

Grayscale intensity is another useful feature which can be used for object tracking. Instead of having three separate channels such as the RGB color space, grayscale only has one channel. **Equation 25** is used to convert an image from a RGB color space to grayscale. The result of using **Equation 25** is shown in **Figure 60**. Once the image has been converted to grayscale, the process of determining the feature parameters is exactly the same as the RGB color space.

Figure 60 - Conversion of RGB Image to Grayscale

The grayscale feature proved to be somewhat more robust than the RGB feature class. Using the same methodology as the RGB feature class, a bin size of 30 was deemed sufficient. The RGB threshold of 0.25 was kept the same for the grayscale feature class.

3. HSV CLASS

The Hue, Saturation and Value (HSV) color space is also useful for computer vision applications. The HSV color space separates the intensity (Value) of the image from two parameters encoding the image's *chromaticity* (Hue and Saturation). This color space is shown in **Figure 61.**

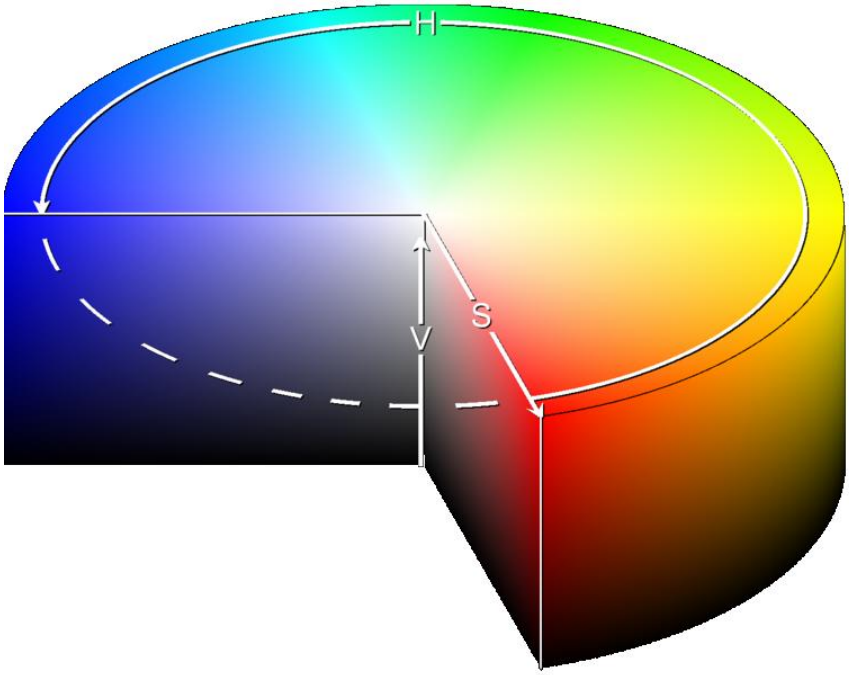

Figure 61 - Visual Representation of the HSV Color Space

The HSV color space was conceived to find a color representation that is more conceptually available to humans [33]. In the RGB color space, humans have a difficult time visualizing which combination of Red, Green, and Blue values must be chosen to develop a particular color. Unless dealing with the three primary colors (Red, Green, Blue), choosing the correct combination simply comes with experience. In the HSV color space, the "color" is determined by the Hue. The "richness" of the color is determined by the Saturation. Finally, the "brightness" of the color is

determined by the Value. For some people, this color representation seems more natural than the RGB color space. The HSV color space sometimes provides better results for computer vision systems. This result is because the program can compensate for lighting (Value), and focus on the parameters of the object's surface [34]. The Hue is defined as an angle between 0 and 360 degrees. The Saturation and Value are both defined as floating points in the range of [0-1].

Because most cameras deliver the image to the user in the RGB color space, they have to first be converted from RGB to HSV. The first step is rescaling the RGB range from integer values of [0-255] into floating point values on the range of [0-1]. This simple conversion is shown in **equation 27.** For example, because the maximum value of the red, green, or blue channel is 255, a red pixel value of 127 becomes 127/255 = 0.498.

$$RGBFloatingPoint = \frac{RGBInteger}{MaxRGBInteger} \quad \textbf{(27)}$$

Now that the RGB values are floating point values on the range of [0-1], the hue can now be calculated. For each pixel, let *max* be the greatest red, green, or blue pixel value, and *min* the least. The calculation of the hue is shown in **equations 28-31.**

$$hue = 0,\ if\ \max = min \quad \textbf{(28)}$$

$$hue = \left(60° \times \frac{g-b}{max-min}\right) mod\ 360°, \qquad if\ \max = red \quad \textbf{(29)}$$

$$hue = 120° + \left(60° \times \frac{b-r}{max-min}\right), \qquad if\ \max = green \quad \textbf{(30)}$$

$$hue = 240° + \left(60° \times \frac{r-g}{max-min}\right), \qquad if\ \max = blue \quad \textbf{(31)}$$

In **equation 29,** the "mod 360" simply means add 360 degrees if the value of the hue is less than zero. Now that the hue has been calculated, the next step is calculating the saturation of the pixel. This saturation is shown in **equations 32 and 33**. Finally, the calculation of the Value is shown in **equation 34**.

$$saturation = 0, \qquad if\ \max = 0 \qquad \textbf{(32)}$$

$$saturation = \frac{\max - min}{max}, \qquad otherwise \qquad \textbf{(33)}$$

$$value = max \qquad \textbf{(34)}$$

Now that the Hue, Saturation and Value parameters have been calculated, their variance needs to be tested. In a manner similar to that of the RGB and Grayscale class, a C++ GUI program was developed to test the variance of HSL color space when objects of a solid color were transformed in space. This program is shown in **Figure 62.**

Once again, the first step was determining a bin size that accounts for normal spatial transformations of the object. One of the main advantages of the HSV color space was its claimed resistance to minor changes in illumination. From testing, the results did not disappoint. Normal spatial transformations of objects caused very minor changes in the Hue and Saturation calculations. Regardless of the object's color, the maximum change in the Hue during normal geometric transformations was 15, on a scale of [0-360]! From these observations, a bin size of 20 should be plenty to cover most normal conditions.

The saturation was also very resistant to changes in the object's orientation. On a floating point scale of [0-1], the maximum variation detected was 0.1. For ease in programming, the

saturation will be rescaled on the integer interval of [0-100]. From this new interval, a bin size of 15 will be chosen to account for most scenarios.

The "Value" in HSV had variations similar to that of the RGB and grayscale class. On a floating point scale of [0-1], the maximum variation was 0.2. The "Value" will also be rescaled on an integer scale of [0-100], and a bin size of 25 will be implemented in the program. Using the same procedures as the previous classes, the L_1 distance for the Hue and Saturation were selected as 0.2. The L_1 distance for the "Value" was selected as 0.25.

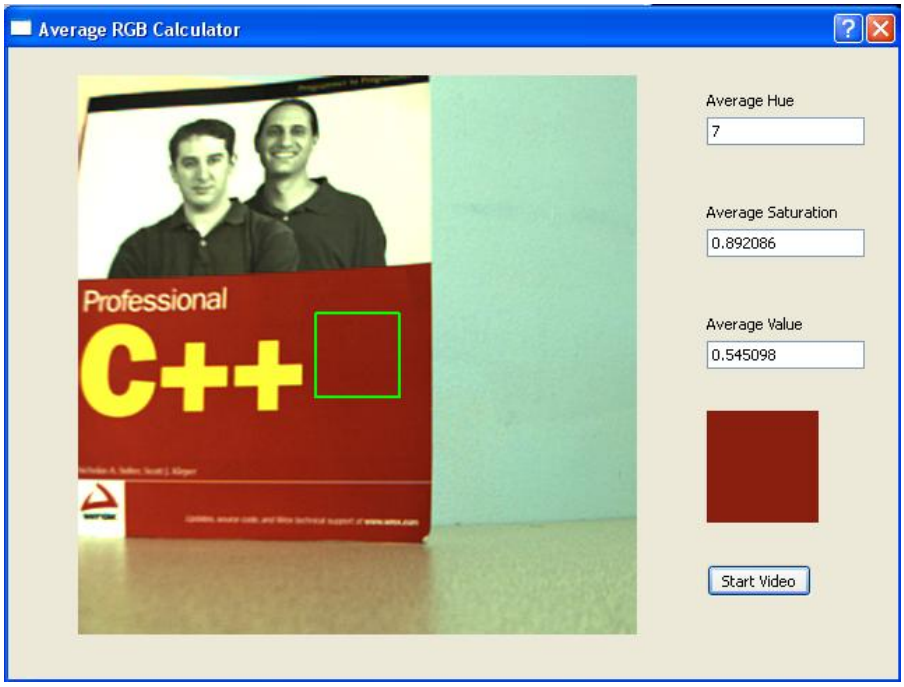

Figure 62 - Program Used to Calculate Average HSV Values

4. TEXTURE CLASS

This fourth feature class will mark a departure away from traditional color spaces. Texture is a feature class that gives information about the spatial arrangements of the color space of an image, rather than the actual colors themselves. The concept of texture is closely related to that of edge detection. A brief overview of texture will now be described.

The first step in determining an image's texture is performing edge detection on that image. There are several operators that are capable of performing edge detection. Looking back at **Figure 31,** one can see the Sobel and Prewitt operators which are used for edge detection. While several edge detection algorithms exist, the Sobel and Prewitt operators are the simplest. As an example, **Figures 63 -65** show the results of the Sobel operator versus the more complicated *Canny and Laplacian* filter.

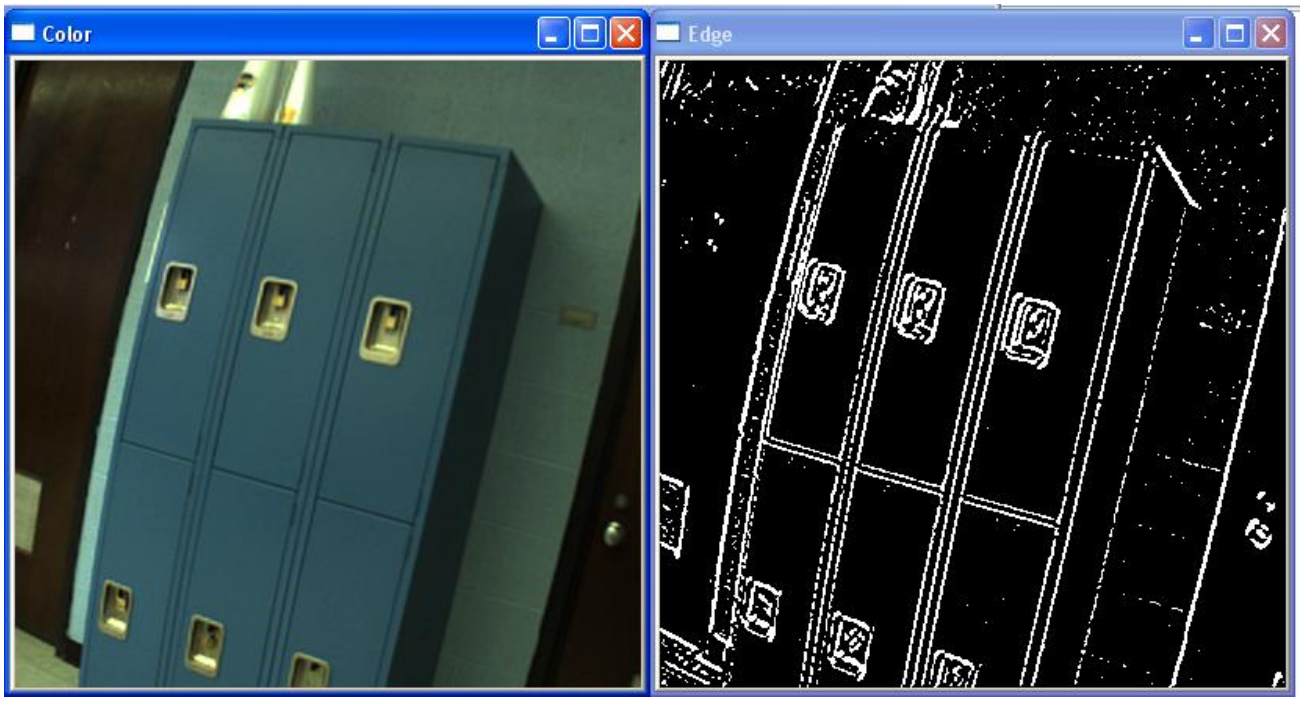

Figure 63 - Results of Sobel Operator

Figure 64 - Results of Canny Operator

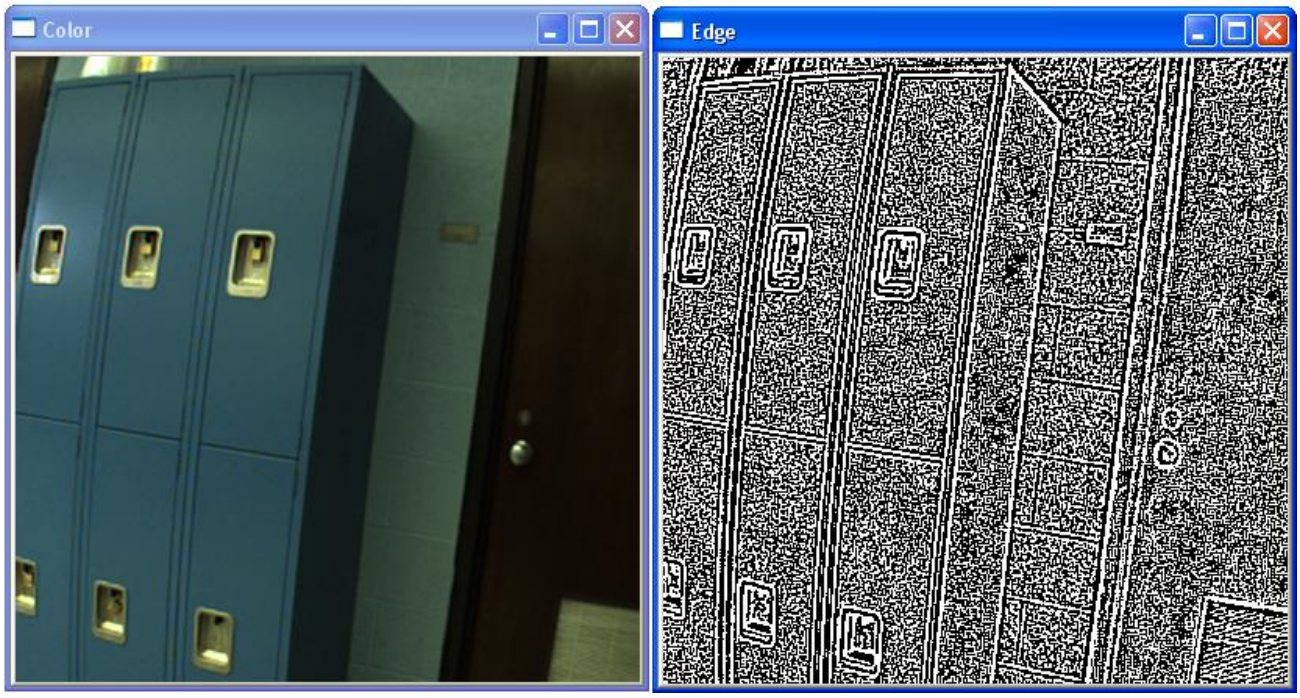

Figure 65 - Results of Laplacian Operator

Looking at the results for both operators, the Laplacian operator tends to have too many clouds to perform any useful sort of texture comparisons. The Sobel and Canny operators provide good separation between areas of high and low texture, while the Laplace operators fill all entities with "noise". Although the Laplace operators certainly have their purpose, the Sobel and Canny operators will be more useful for this texture class. Once edge detection has been performed on the image, the texture can be calculated as shown previously in **Figure 38**.

The texture measure to be used for this class is *edgeness per unit area* as shown in **Figure 38**. In this class, a 3×3 kernel will be moved across the image and the edgeness values will be placed into histograms. This 3×3 kernel was chosen because of concerns related to calculating the bin size. The objects to be tracked vary in size. If the object being tracked becomes very small, a kernel which is too large will not give enough hits on the object to perform accurate calculations. Using a 3×3 kernel will give accurate results for general object tracking applications. Based on this kernel size, the edgeness can take on a maximum of 10 values$\left(\frac{0}{9}\ through\ \frac{9}{9}\right)$. Because of this fact, a histogram with 10 bins will be selected for initial testing. Using the same method as previous classes, an L_1 threshold of 0.20 was selected.

5. IMAGE WEIGHT

The overall image weight is another simple feature which can be used for object classification. Like the "texture" feature, the weight of the image is strongly based on edge detection. The normalized weight of an image $w \in [0,1]$ is simply the sum of the edge pixels divided by the total number of pixels in the image.

Unlike previous classes, the distance measure for the image weight will not be based on the L_1 threshold. The image weight is a simple number describing an image; therefore histograms methods do not apply. To measure the separation between image weights, a simple percent difference will suffice **(equation 35).**

$$percent\ difference = \frac{Observed - Expected}{Expected} \quad \textbf{(35)}$$

Using this simple distance measure, empirical variance testing showed that a percent difference below 25% would qualify this feature as good.

6. IMAGE MOMENT OF INERTIA

Moment of Inertia (MOI) is the polar moment of inertia of an image. Similarly to the image weight class, MOI is based on edge detection. Most digital images are stored in raster orientation with the origin of the image being the top left corner. However, to ease the implementation process, the image's center 'c' will be chosen as the origin **(Figure 66)**.

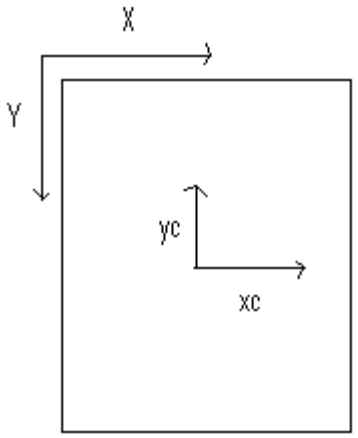

Figure 66 - Image Origin for Calculating MOI

The first step in calculating an image's MOI is to perform edge detection as with the previous two classes. Two methods for calculating a moment of inertia are treating the entity as a rigid body and treating the entity as a summation of discreet points. Both methods will be used to calculate a normalized MOI. This normalized MOI is needed so that the results can be used on images of varying sizes.

Determining the MOI consists of two steps: calculating the "raw" MOI, and normalizing the MOI on an interval of$M \in [0,1]$. To calculate the raw MOI, the image will be treated as a cloud of points **(Figure 67)**. The raw MOI is then calculated by **equation 36** where n is the number of edges in the image and m is the mass. Because the mass 'm' is a simple pixel, it will be set to unity. The moment of inertia now becomes a summation of the squared distances from an edge to the origin.

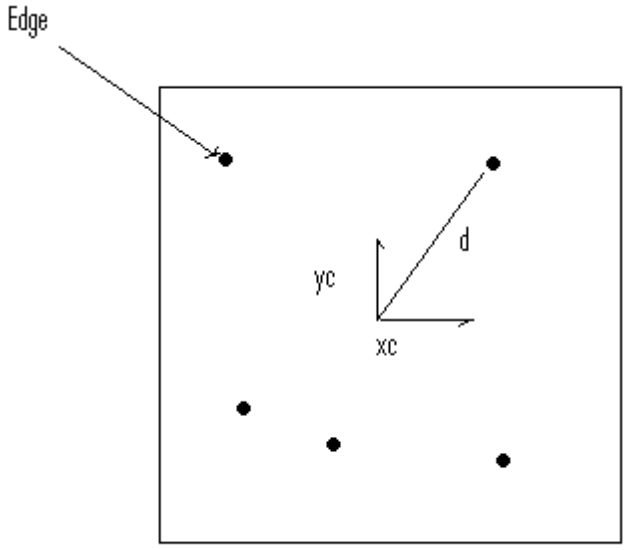

Figure 67 - Calculating the Image MOI from Edges

$$M_c = \sum_{i=1}^{n} m \times d_i^2 = \sum_{i=1}^{n} d_i^2 \quad \textbf{(36)}$$

Once the raw MOI has been calculated by treating the image as a cloud of points, the result must be normalized. To normalize the MOI, the maximum value of the MOI must be determined. This task is accomplished by treating the image as a rigid body. Treating the image as a rigid body is essentially the same as assuming every pixel is an edge. This simplification allows the use of **Equation 37** to calculate the maximum MOI of an image about its center. The normalized moment of inertia is calculated by dividing the point cloud MOI by its maximum value (**Equation 38**). Using the same percent difference method as the image weight class, testing showed that a percent difference below 25% would be qualify this feature as good.

$$M_{c,max} = \frac{bh}{12}\left(b^2 + h^2\right) \quad \textbf{(37)}$$

$$M_{c,normalize} = \frac{M_c}{M_{c,max}} \quad \textbf{(38)}$$

7. – 10. HISTOGRAM MODAL ANALYSIS

Histogram modal analysis is another simple feature class to determine whether a test image belongs to the same category as the training images. With the previous histogram based feature classes (RGB, HSV, Texture, Grayscale), the distances between histograms was calculated using the L_1 measure **(equation 24)**. The L_1 measure compares the difference between each bin in two separate histograms.

Histogram modal analysis is a much more relaxed method of comparing two histograms. Instead of comparing each bin in two separate histograms, we simply compare their histograms modes. For example, the red channel histogram of the beverage container in **Figure 68** is shown in **Figure 69**.

Figure 68 - Beverage Container

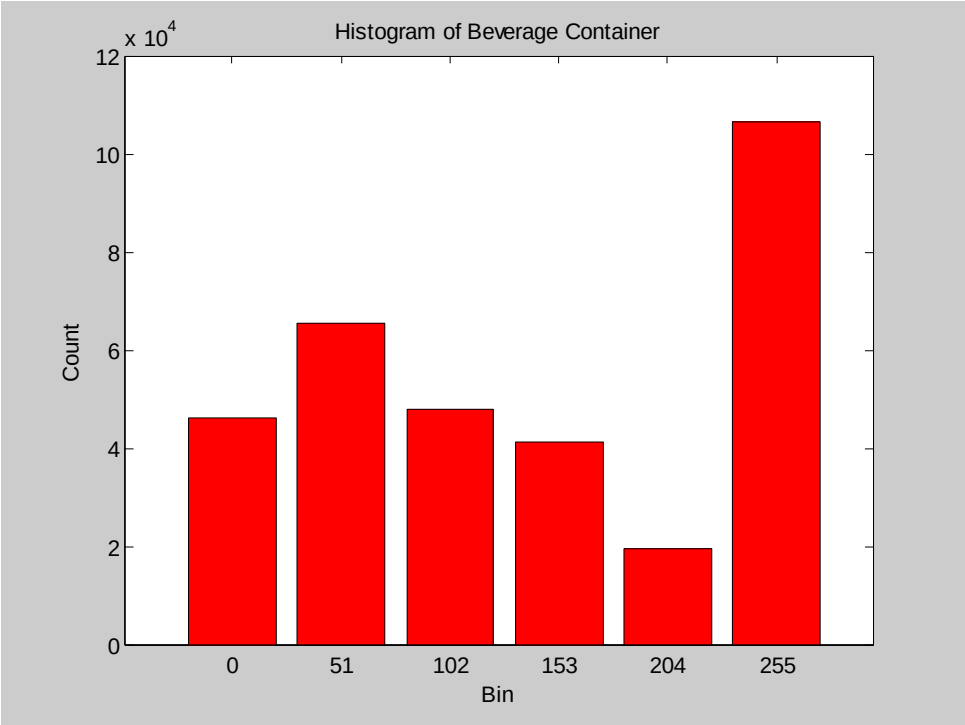

Figure 69- Histogram of Red Channel for Beverage Container

From observation, it can be seen that the red channel histogram's mode is the bin that contains the values of 255. To determine whether the feature class is acceptable for finding objects in future images, this mode is compared to the mode of any future images. If the histograms contain the same mode, the feature is classified as good. If the histograms contain different modes, the feature class is rejected.

Unlike previous feature classes, there is no L_1 distance or percent difference. Whether different histograms have the same modes is either *true* or *false.* As stated, this modal analysis is a relaxed method of histogram comparison that will be used for the RGB, HSV, Texture, and Grayscale Feature classes.

11. NUMBER OF CIRCLES

The eleventh feature class that will be implemented is the *number of circles* presented in an image. This circle detection algorithm will be designed using Hough transform principles. The Hough transform is a feature extraction method that uses an accumulator array to find instances of objects by a "voting" procedure. The Hough transform is used mainly for finding lines or line segments, but the transform can also be applied to other well known shapes [35].

As stated, the Hough transform requires an accumulator array. The dimensions of this accumulator array are determined by the number of parameters of the curves being determined. The equation of a line is shown in **equation 39**. Because **equation 39** cannot be used for representing vertical lines and the coordinates are in Cartesian space, **equation 40** will be used to represent a line in raster coordinates. These values are shown in **Figure 70**.

$$y = mx + b \quad \textbf{(39)}$$

$$d = ccos\theta - rsin\theta \quad \textbf{(40)}$$

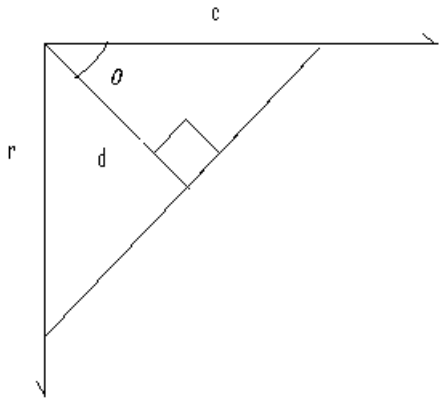

Figure 70 - Parameters of a Generic Circle

The first step in the Hough line finder is edge detection. This edge detection can be performed using any of the previous edge detection methods such as the Sobel, Prewitt, or Canny operator. The next step in the Hough method is analyzing the edge image for the presence of lines. To determine the presence of a line, each pixel and its 8 neighbors are analyzed for edges as shown in **Figure 71.**

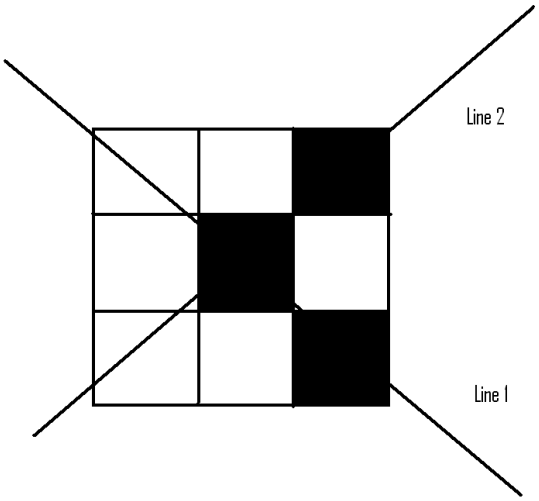

Figure 71 - Line Detection for Hough Transform

In this example, two possible lines have been detected. The distance 'd' and angle $'\theta'$ are then calculated. This procedure is repeated for each pixel in the image and the results are stored into an accumulator such as **Figure 72**. If the number of similar lines reaches a predefined threshold, the Hough method classifies that line as a hit. Once a line has been classified as a hit, post processing of the line's distance and angle measurements can be performed to determine the length of the actual line segment.

	Distance	Angle (degrees)
Line 1	34	12
Line 2	12	3
Line 3	4	78
Line 4	34	12
Line 5	34	12
Line 6	2	9
Line 7		
Line 8	12	3
Line 9	12	3
Line 10	87	176

Figure 72 - Generic Accumulator

A similar procedure can also be used to find circles in an image. The parametric equations of circle are shown in **equations 41 and 42.**

$$r = r_o + dsin\theta \quad \textbf{(41)}$$

$$c = c_o - dcos\theta \quad \textbf{(42)}$$

By observing **equations 41 and 42**, one can see that a circle consists of three unknowns, r_o, c_o, and d. In order to compensate for this extra variable, the radius 'd' will be added to the accumulator. To find the center of a circle for a specified radius, the gradient at each pixel point must first be calculated **(Figure 73)**. The gradient of a pixel is the direction of the largest intensity increase and is shown by **equation 43.**

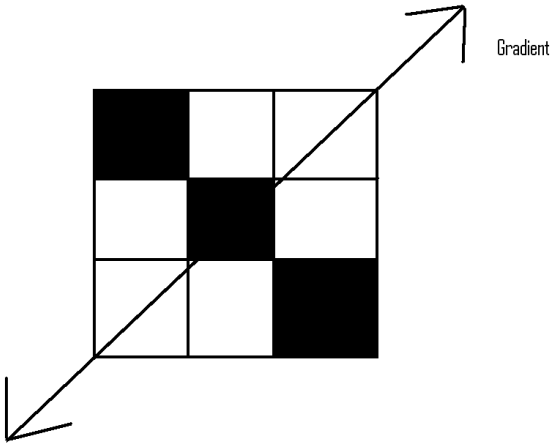

Figure 73 - Gradient at a Single Pixel

$$\nabla I(r,c) = \frac{\Delta I}{\Delta r}\vec{r} + \frac{\Delta I}{\Delta c}\vec{c} \quad \textbf{(43)}$$

If a point lies on a circle, then the gradient points to the center of that circle as shown in **Figure 74.** Once the gradient at each pixel has been calculated, the procedure follows that of the Hough line finder. The values r_o,c_o, and d are all stored into an accumulator. If the values pass a predefined threshold, then the curve is classified as a circle. The results of this procedure are shown in **Figure 75**.

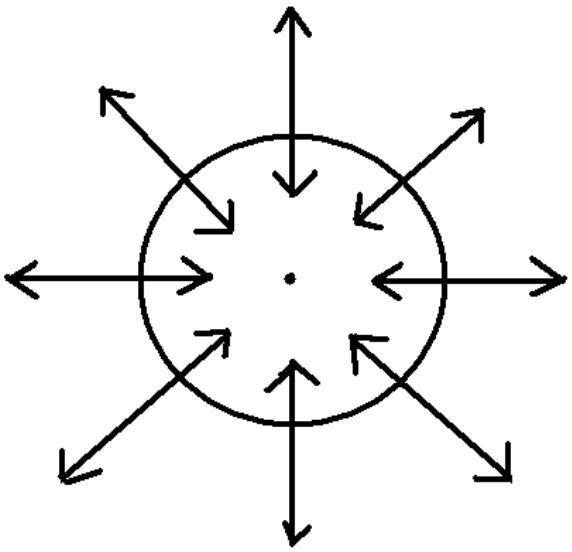

Figure 74 - Gradient of a Circle

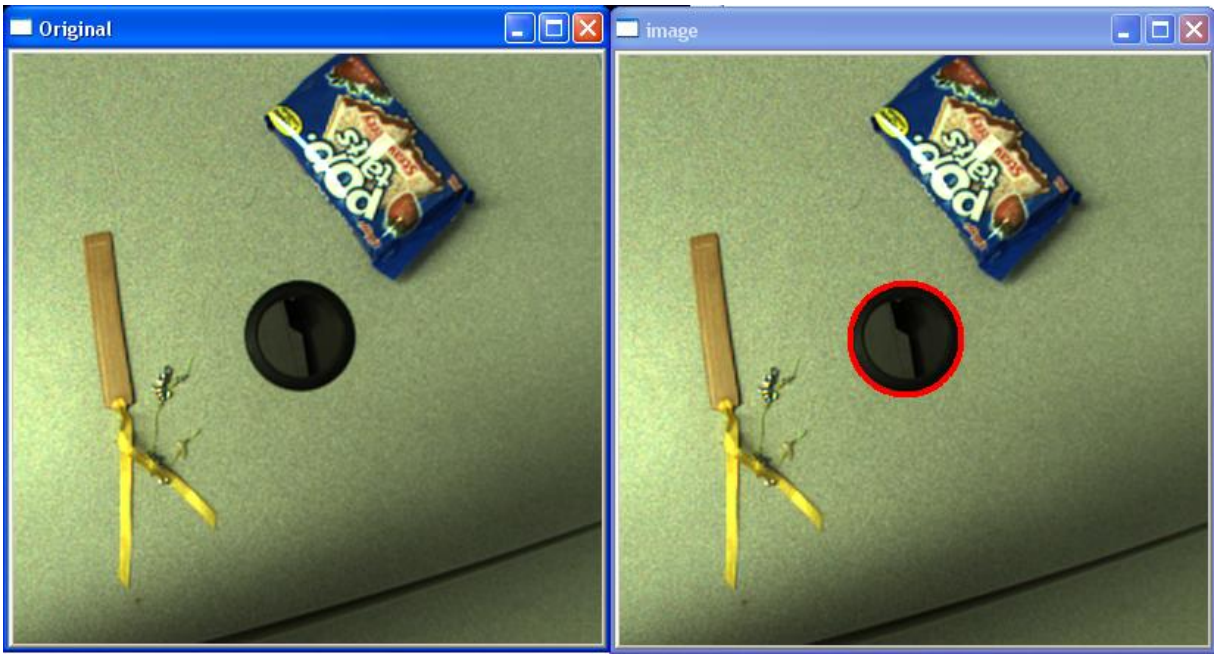

Figure 75 - Results of Circle Detection

12. NUMBER OF RECTANGLES

To calculate the number of rectangles in an image, the image processing technique of *connected component labeling* will be implemented. Connected component labeling groups pixels into classes based on pixel connectivity. Connected component labeling is usually performed on binary Images**.** To obtain a binary image, the color image is simply converted to grayscale using **equation 25**. Once the image has been converted to grayscale, each pixel is then compared to an intensity threshold. If the pixel falls below this intensity threshold, its value is set to 0, otherwise the pixel is set to 1.

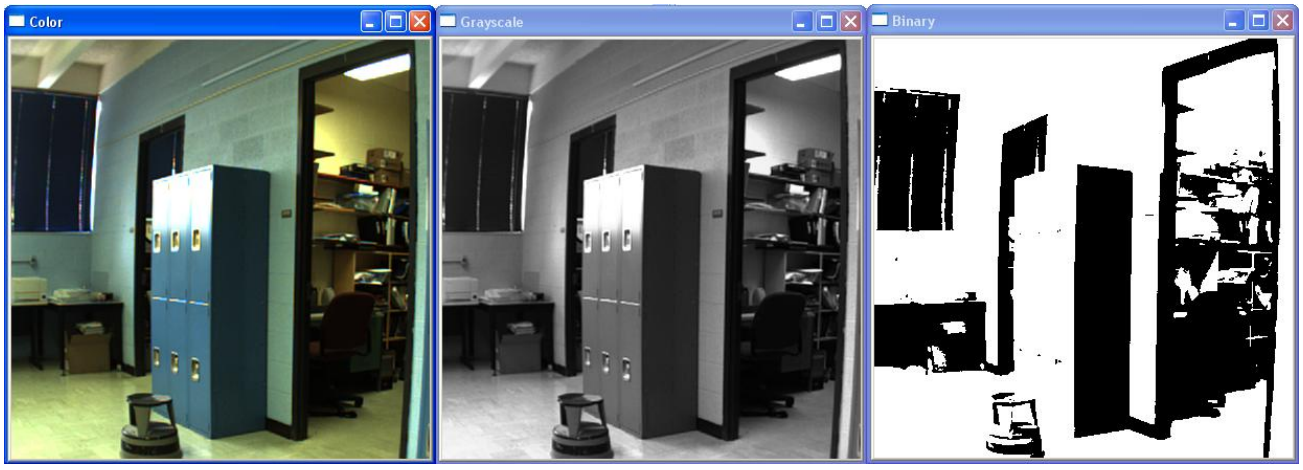

Figure 76 - Binary Image Representation

Once the image has been converted into its binary representation, connected component labeling is performed. An excellent description of connected component labeling can be found at *Image Analysis – Connected Component Labeling* [36]. Connected component scans an image, pixel by pixel, from top-left corner to bottom-right corner. When a pixel of value 1 is encountered

the algorithm compares the pixel to its left, right, and two diagonal neighbors (**see figure 77**). The pixel of interest p is compared to the following rules.

- If all four neighbors have a value of 0, assign a new label to *p*, else
- if only one neighbor has *V={1}*, assign its label to *p*, else
- if more than one of the neighbors have *V={1}*, assign one of the labels to *p* and make a note of the equivalences.

Afterwards, pixels with the same label are all assigned to the same class.

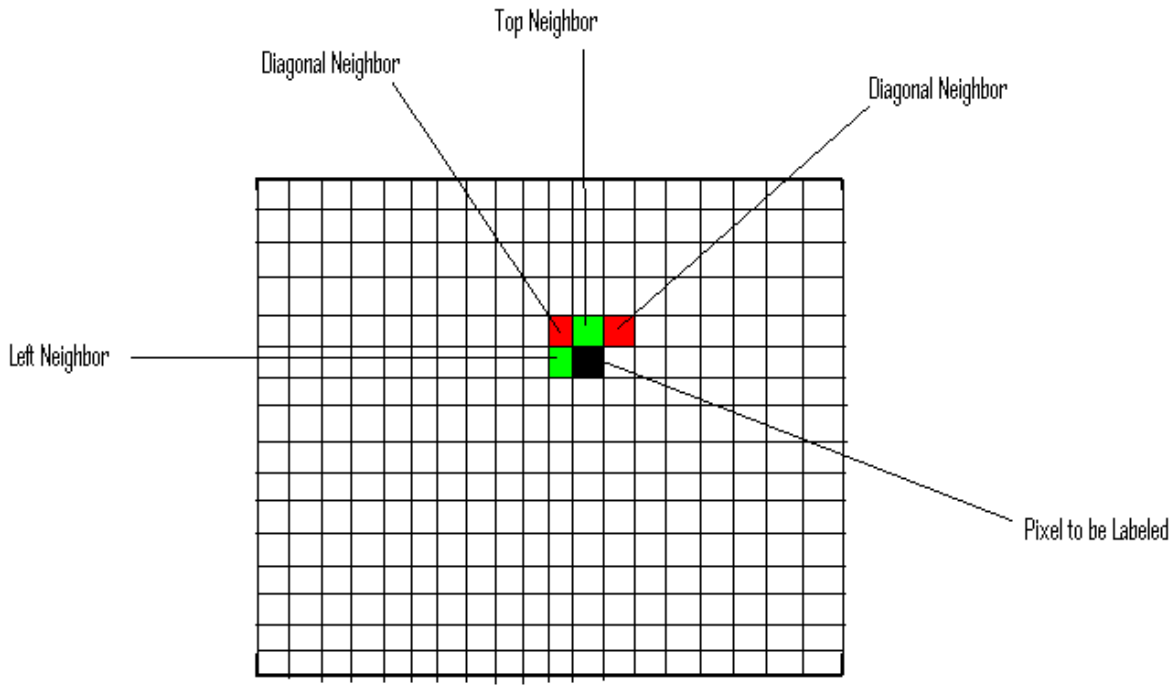

Figure 77 - Neighbors of Pixel to be labeled

The results of connected component labeling are shown in **Figure 78.** Now that connected components have been identified in the image, the next step is identifying which contours in the image represent rectangles.

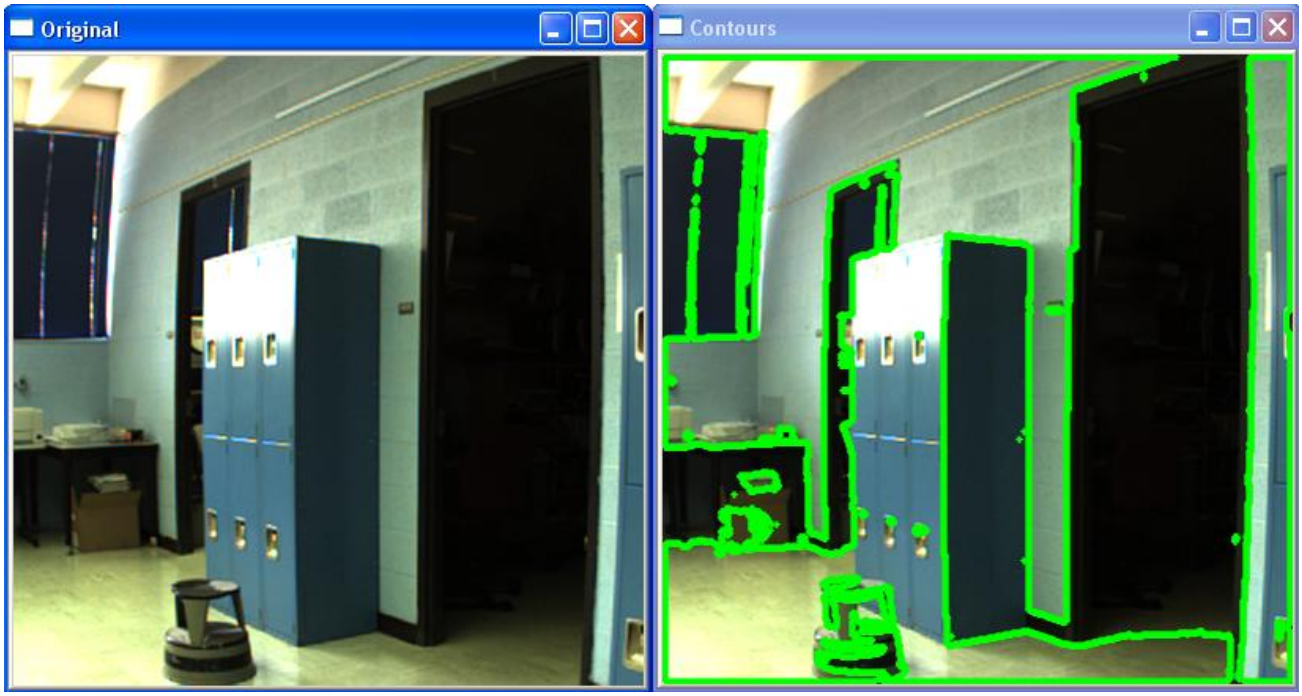

Figure 78 - Results of Connected Component Labeling

To simplify the process of identifying contours as rectangles, Intel's Open Source Computer Vision Library (OpenCV) will be used. This library has a function that approximates polygonal contours. This function implements the Douglas-Pecker algorithm for finding a simple polygon that approximates the original within a specified tolerance. The details of the algorithm are beyond the scope of this research; for more information on this algorithm one should read *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature* [37].

Once all the contours have been approximated as polygons, the polygons must be classified as rectangles. In order for a polygon to be classified as a rectangle, it must pass the following criteria.

- Each polygon must have 4 vertices
- Each vertice must concave (curve inward)
- Each vertice must have approximately a 90 degree angle

Once these restrictions have been placed on the list of approximated polygons, only the rectangles in the image should remain. The results of this rectangle finding are shown in **Figure 78**.

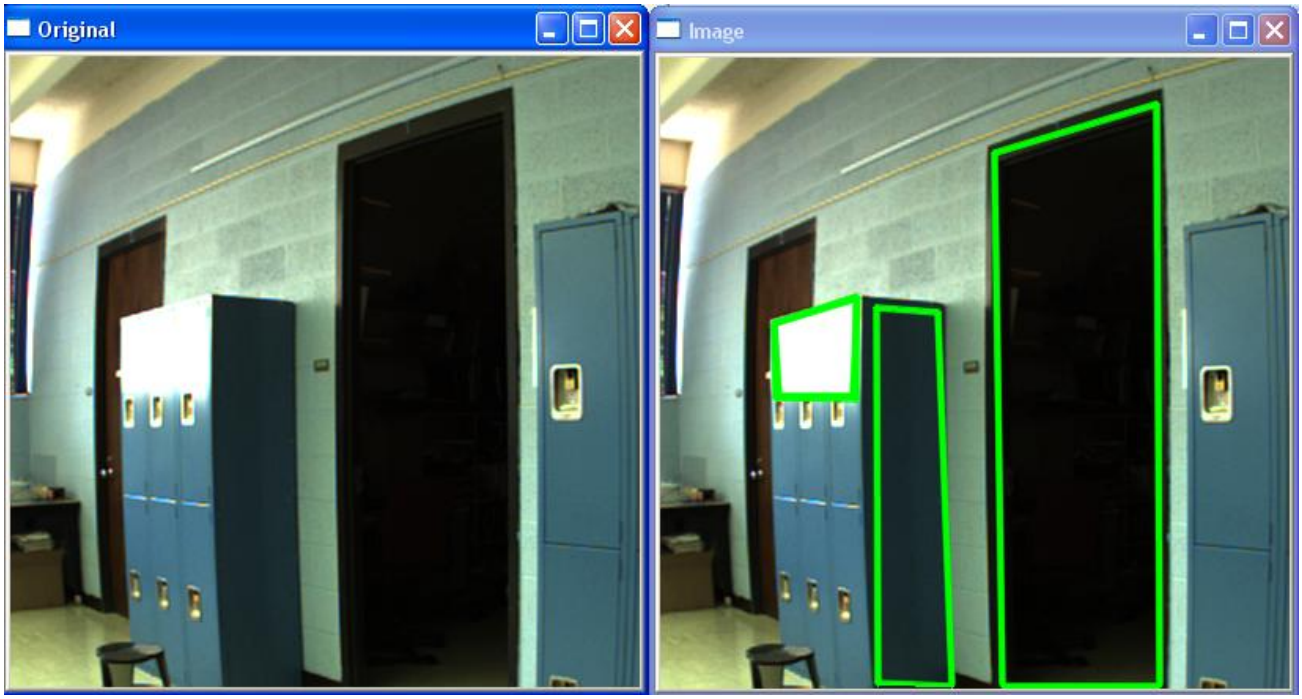

Figure 79 - Rectangle Finding in an Image

With the conclusion of this chapter, the development of the features classes has been illustrated. As stated, not all the feature classes were able to fit into the framework of the *methodology* section; however any deviations from the *methodology* section were described in detail. The next chapter will demonstrate how an object-oriented design approach will be used for implementation of these feature classes into the overall program assembly. This object-oriented approach will allows easy modification of the current feature classes and easy addition of any future feature classes.

CHAPTER 7 – APPLICATION ASSEMBLY

As stated in the *abstract* section of this dissertation, an object oriented approach will be used for the overall program assembly. By incorporating an object oriented design (OOD) philosophy, different portions of the application can be modified without modifying or damaging the entire program. Moreover, an OOD design philosophy allows for easy future expansion of feature classes. This chapter will give an introduction to object oriented programming and will show how the feature classes will be implemented using this programming philosophy.

"*Writing a program without a design is like building a house without blueprints"*

-Professional C++ [38]

Program design is the single most important step in application development. However most individual program developers, even professionals breeze over this step as if it's just a hassle. Programmers have a tendency of coming up with a rough idea (usually in their head) of how a program should work and then immediately begin coding. The typical time distribution for most individual programmers is 20% design and 80% coding. This neglect of the design phase only works for the most simplest of programs. Without adequate design, the programmer will almost always run into problems that could have easily been avoided. Unfortunately, these problems usually show themselves when the programmer is nearly done with the program.

To illustrate the importance of program design, an example will be given. One of the benefits of this research was to allow the feature classes to be easily modified or extended long after the original designer of the application has left. The graphical user interface (GUI) portion of this program was done using Trolltech's QT4 application development platform. However, if the functionality of each feature class was intertwined with the GUI code, every subsequent user of the application would also have to be an expert with QT4.

With this approach, anyone who wanted to modify or extend the features classes would also be forced to modify preexisting GUI code. Not only would this be annoying to the end user, it would be dangerous. By changing the GUI code, he/she runs the risk of breaking a previously working application.

The author of this dissertation is not foolish enough to believe everyone loves developing GUI's using QT4. Several people who work with the author prefer using Microsoft's very popular C# programming language. By planning ahead, the features classes can be easily extracted from the application and placed in any GUI development environment, thus allowing the end programmer to use whatever GUI language in which he/she is most comfortable.

Once good programming design has been performed, there are two main programming practices that exist today, *procedural* and object-oriented. Procedural code (typically done using 'C') revolves around data, and functions that act on that data. Procedural programming asks the questions – "*What does this program do?*" By answering this question and designing the program as a series of steps, one is designing procedurally.

While procedural programming works well if a program follows a sequential series of steps, it is plagued with problems in larger applications. The first problem is that the data and the functions that act on that data are in different locations. If a function changes its arguments, then the data on which the function operates must also change its structure. Moreover, because the data is separate, the data is accessible to any part of the program. This lack of data privacy is a major source of errors and security concerns in many programs.

The second problem with procedural programming is breaking pre-existing code. Because procedural code consists of functions that operate on data, the function must be able to handle whatever data type is passed to it. Because of this, any future modification to the program involves possibly changing code that was previously working.

The third problem with procedural programming is redundant code. Either a function must be able to handle every data type it's given, or a separate function must be created for each data type. If a new function is created, the program calling that function must also be changed. More code and more changes imply more possibilities for error.

The object-oriented (C++, C#, Java, and others) approach works around these problems by treating the program as a set of objects that interact with each other. Instead of separate functions and data, both are contained in the same location. Moreover, this object-oriented approach allows for easy future expansion of the program. Instead of asking "What does this program do?" the question is instead "What Objects are to be modeled."

Object oriented programming has three main advantages over procedural programming.

1. Encapsulation
2. Inheritance
3. Polymorphism

Encapsulation consists of keeping the data and functions (methods) that act on that data in one location called a class. By putting the data and methods in a single class, this class can be treated as a separate entity. By treating the program as a set of standalone classes, different parts of the program can be modified without unintentionally breaking something else.

Inheritance is the process of creating a superclass from a set of base classes. For example, assume a program is created that mimics farm animals. These animals consist of dogs, cats, pigs, cows and horses. Each of these animals shares several of the same behaviors such as the following:

1. Eat
2. Walk
3. Run
4. Sleep

Because all the animals share these same behaviors, it wouldn't make sense to program different methods for each animal as shown in **Figure 80**. Instead, a superclass will be created that contains the common functionality of each animal as shown in **Figure 81**. By abstracting this common functionality to the superclass, a tremendous amount of programming time is avoided. Furthermore, if someone wants to add a new animal to the program, he/she can simply inherit the common functionality from the superclass and avoid typing the code themselves.

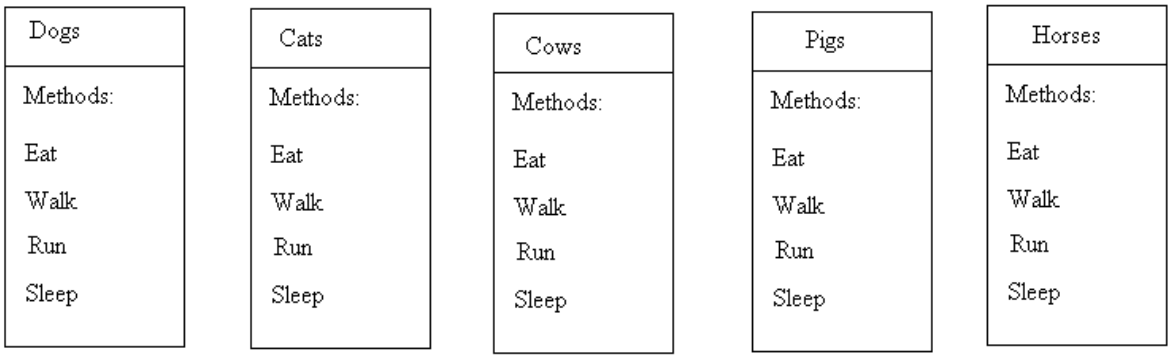

Figure 80 - Design of Animals without Inheritance

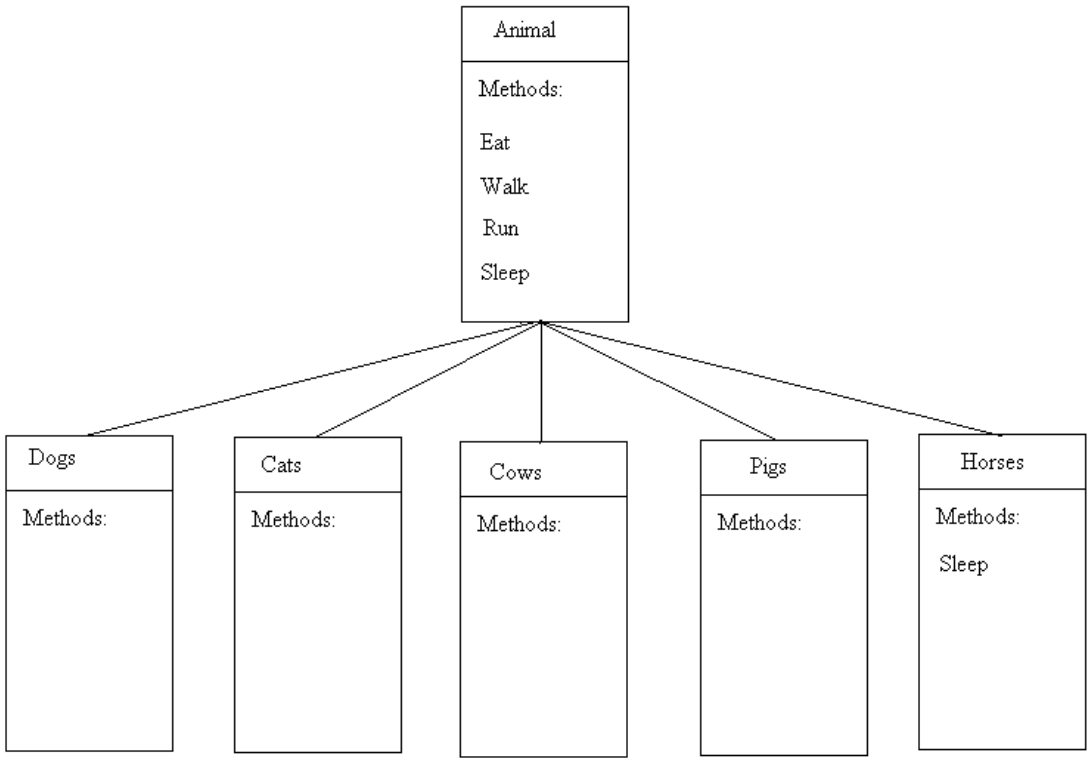

Figure 81 - Design of Animals with Inheritance

The final benefit of an object oriented design approach is polymorphism. Polymorphism allows a subclass to override the functionality of its superclass. In this example of farm animals all the animals usually sleep lying down, except the horses. Because of this, we can simply supersede the generic Animal's sleep method with a method particular to that of a horse. Unlike procedural programming, this modification can be performed without affecting any other part of the program.

A technique of polymorphism that is used for programs which will undergo continuous expansion is programming with *interfaces.* An interface is a superclass that contains method prototypes, but doesn't contain any underlying functionality. One might ask why interfaces are ever needed; wasn't the goal of object oriented programming to reduce redundant code?

Interfaces are useful when several subclasses contain the same methods, but implement each method differently. As an example, consider the twelve feature classes which were presented in the previous chapter. Although each feature class calculates the distance between two or more images, each feature class performs this operation in a different manner. Any future feature classes will also calculate the distance between images using a different approach.

Interfaces allow the programmer to design his/her program for future expandability. By designing the program using a generic "feature" superclass, any subsequent feature classes can be implemented with ease. As long as the subclass overrides the method prototypes in the superclass, expansion can be done without modifying any other part of the program.

This feature class interface is shown in **Figure 82**. As shown, the interface has two methods that are used in the program, "IsFeatureGood" and "ProcessImage". The "IsFeatureGood" method checks whether the feature will be appropriate for finding the object in future images. This method is called during the training session of the program. The second method "ProcessImage", is called when the program is looking for the object in future images. Therefore, as long as each subsequent feature class implements these two methods contained in the interface, the feature class can then easily be added into the overall program.

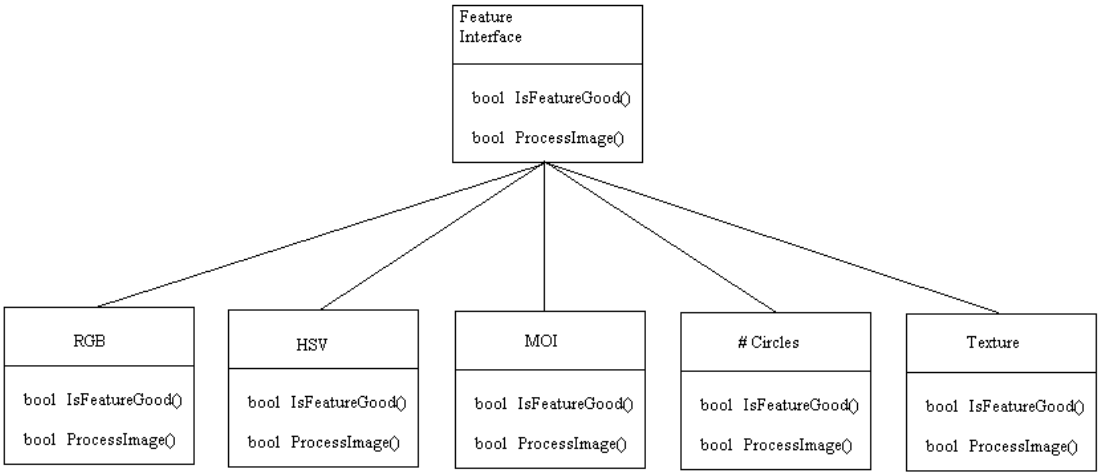

Figure 82 - Programming with Interfaces

Two programming libraries will be used in the implementation of this program. The first library is Intel's Open Source Computer Vision Library. This library consists of a huge set of low-level functions designed to make the computer vision engineer's life much easier. Although the library is written purely in 'C', a C++ wrapper will be placed around these functions to make them adhere to the object oriented framework.

The second library used in this program is Trolltech's QT4. QT4 is a completely object-oriented, C++, cross-platform application development framework. QT4 was chosen because C++ programs tend to operate much faster than other languages. QT4 was also chosen because all of the pre-existing code was already in C++, therefore adding another programming language would add unnecessary complications. As stated however, if one wishes to move the feature classes to another language, he/she will be able to do so relatively easily.

The fully working program will now be demonstrated. This program will be used to determine which set of features are good for tracking the QT4 book. Next, based on which features the program selects as good, the program will then find the object of interest in future images.

The fully working program is shown in **Figure 83**. In this figure, the first image of the QT4 book is selected by the user.

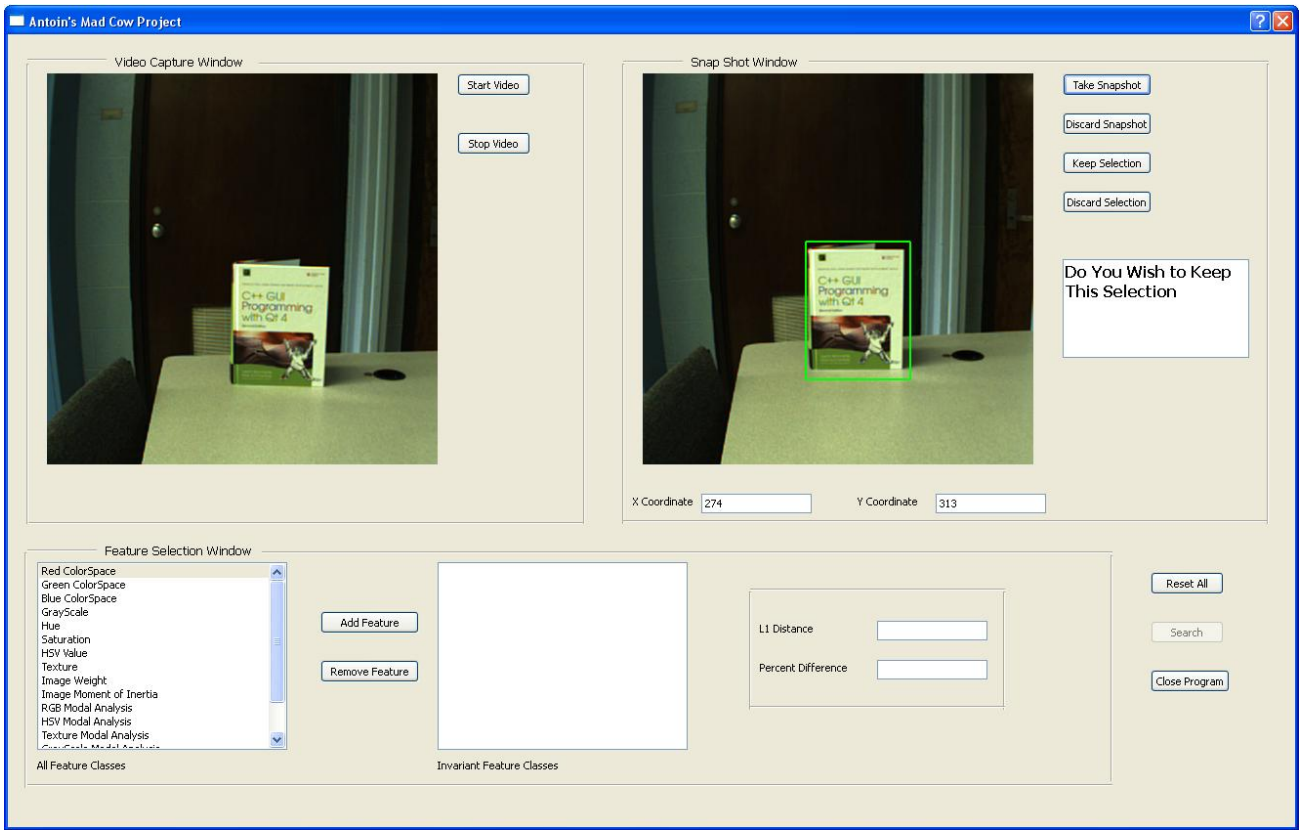

Figure 83 - First Selection of Object of Interest (QT4 Book)

Next, a rotated view of the QT4 book is selected. This selection is shown in **figure 84**.

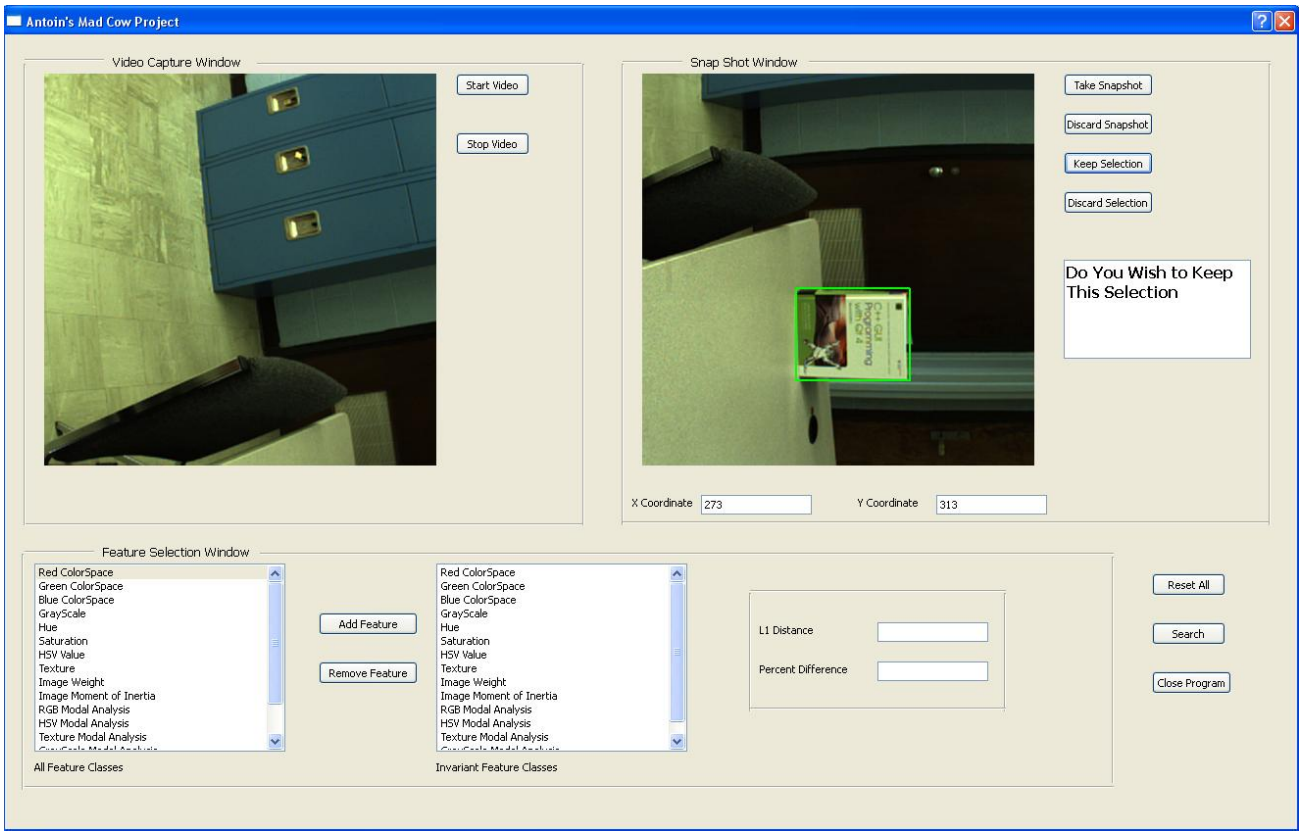

Figure 84 - Second Selection of Object of Interest (QT4 Book)

Based on these two selections, the features which the program deemed as good are shown in **figure 85.** Because the images were nearly identical except for the rotations, the majority of the feature classes were chosen as good. As one can see, the feature classes of circles and rectangles failed. Unfortunately, these feature classes which consist of primitive shapes usually fail under most real-world conditions as near perfect primitive shapes rarely exist in nature.

ALL FEATURE CLASSES	GOOD FEATURE CLASSES
RGB	RGB
GRAYSCALE	GRAYSCALE
HSV	HSV
TEXTURE	TEXTURE
IMAGE WEIGHT	IMAGE WEIGHT
IMAGE MOI	IMAGE MOI
GRAYSCALE MODAL	GRAYSCALE MODAL
RGB MODAL	RGB MODAL
HSV MODAL	HSV MODAL
TEXTURE MODAL	TEXTURE MODAL
# CIRCLES	
# SQUARES	

Figure 85 - Selected Features

The user has the option to add or remove feature classes based on his/her expert knowledge of the situation. Once the user is satisfied with the feature classes, he/she then presses the "search" button and the program will then use the good feature classes to find the object in future images. Finding the QT4 book in future images is shown in **figure 86**.

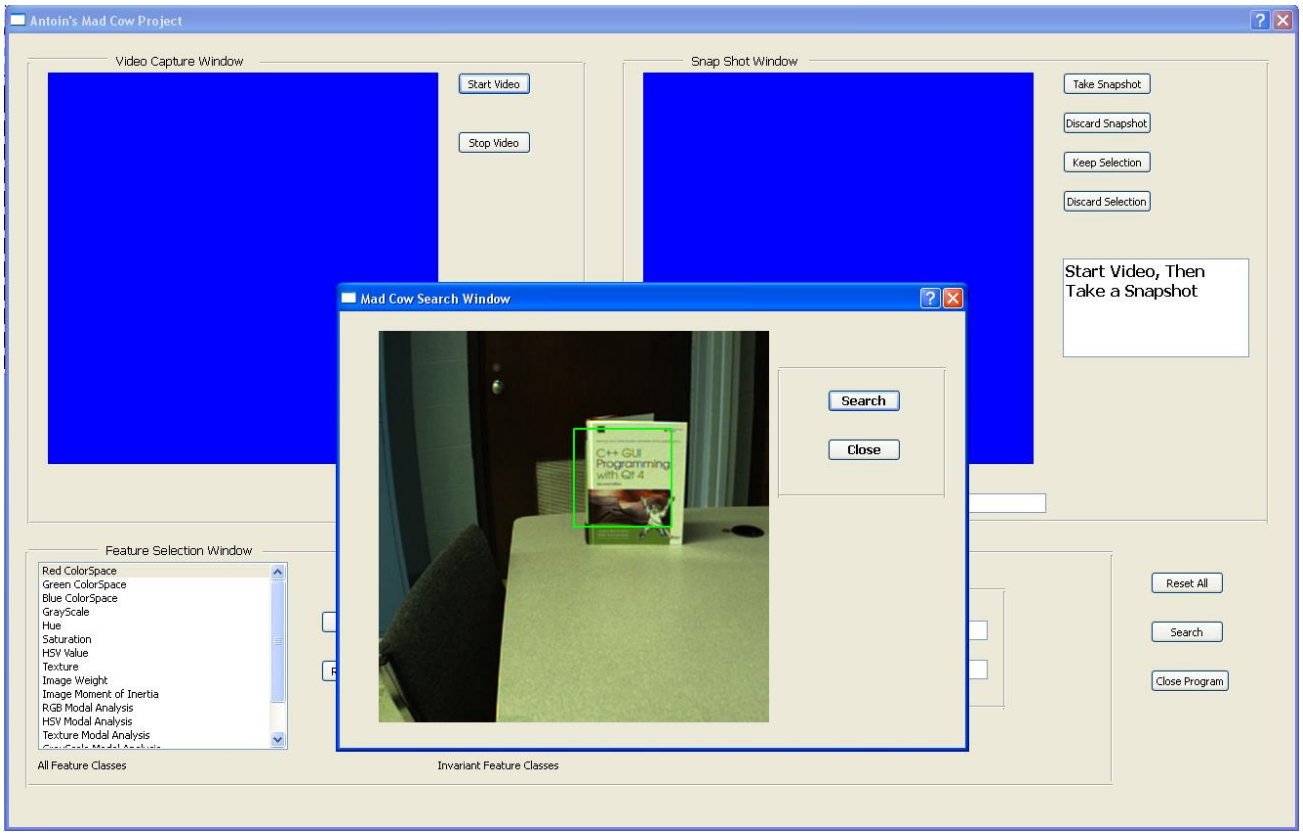

Figure 86 - Finding Object of Interest (QT4 Book) in Future Images

With the conclusion of this chapter, one should understand why an object oriented program philosophy is so beneficial to the development of this program. Using this object oriented approach, the development of the individual feature classes and how each feature class is implemented in the overall program has been demonstrated. The fully working program has also been used to track an object. The next chapter will compare this program against two popular methods of object detection as described in the *methodology* section of this dissertation.

CHAPTER 8 – TESTING AND COMPARISON

This chapter compares this new feature selection method with two very popular methods of object detection. These two popular methods consist of image template matching and a simple statistical based neural network which is built into Intel's OpenCV library. This chapter will give a brief introduction to these well known methods and then show how these methods compare to the new application.

The first method to be discussed is image template matching. Template matching is the process of finding small parts of a large image (**figure 87)** which match a pre-specified template. The only requirement is that the template (**figure 88)** must be smaller than the image being searched.

Figure 87 - Entire Image to be Searched

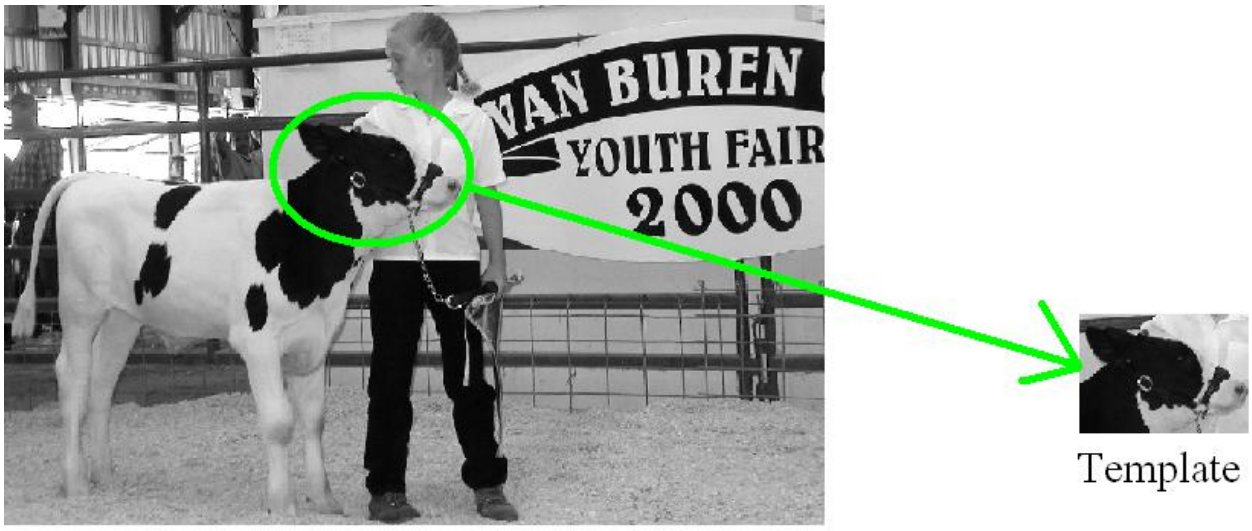

Figure 88 - Example of Template

To find the template in the larger image, the template is usually compared to the larger image using the sum of absolute differences SAD method (**equation 44**). The template image is simply moved across the larger image and the sum of absolute differences is repeatedly computed. If the SAD is below a certain threshold, then the method declares a match has been found in the larger image.

$$SAD(x,y) = \sum_{i=0}^{TRows} \sum_{j=0}^{TColumns} |diff(x+i, y+i)| \quad (44)$$

The template matching program is fairly simple. The user selects which part of the image he/she wants to use as a template. If the user is happy with the template, he/she then clicks search and the program attempts to find the template in future images.

In this example, the goal is the tracking of the QT4 book. The video stream is shown in **Figure 89**. Whatever image is inside the central square becomes the template. This template is shown in **figure 90**.

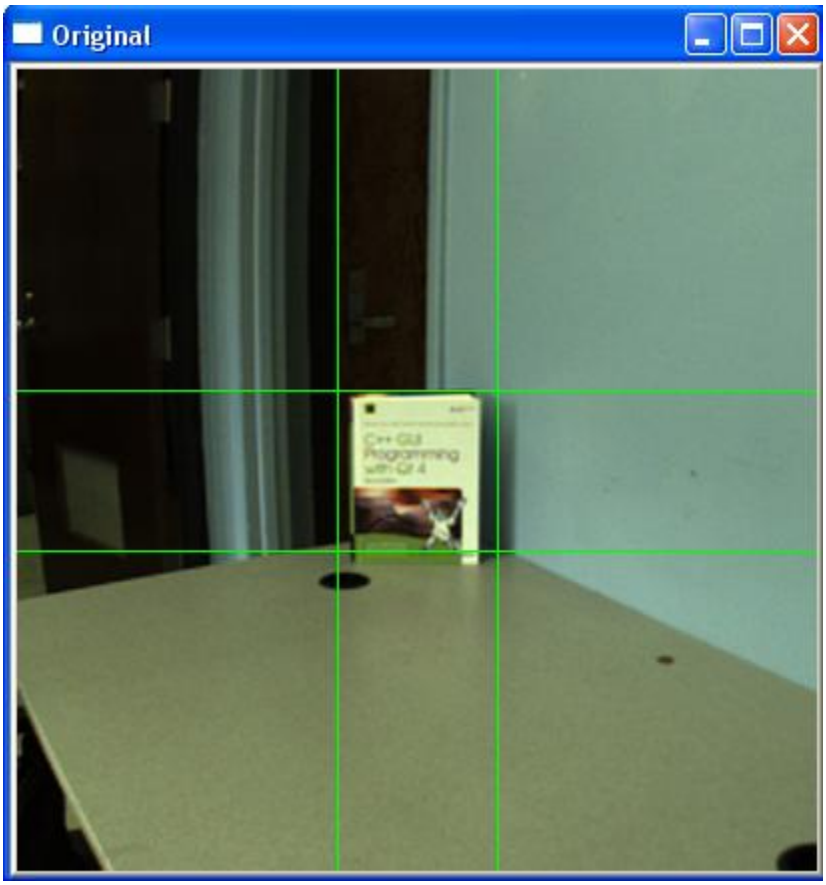

Figure 89 - Image Used to Find Template

Figure 90 - Template Used for Future Image Searches

When the user is satisfied with the template, he/she then tells the program to search future images. The searching of future images is shown in **Figure 91**.

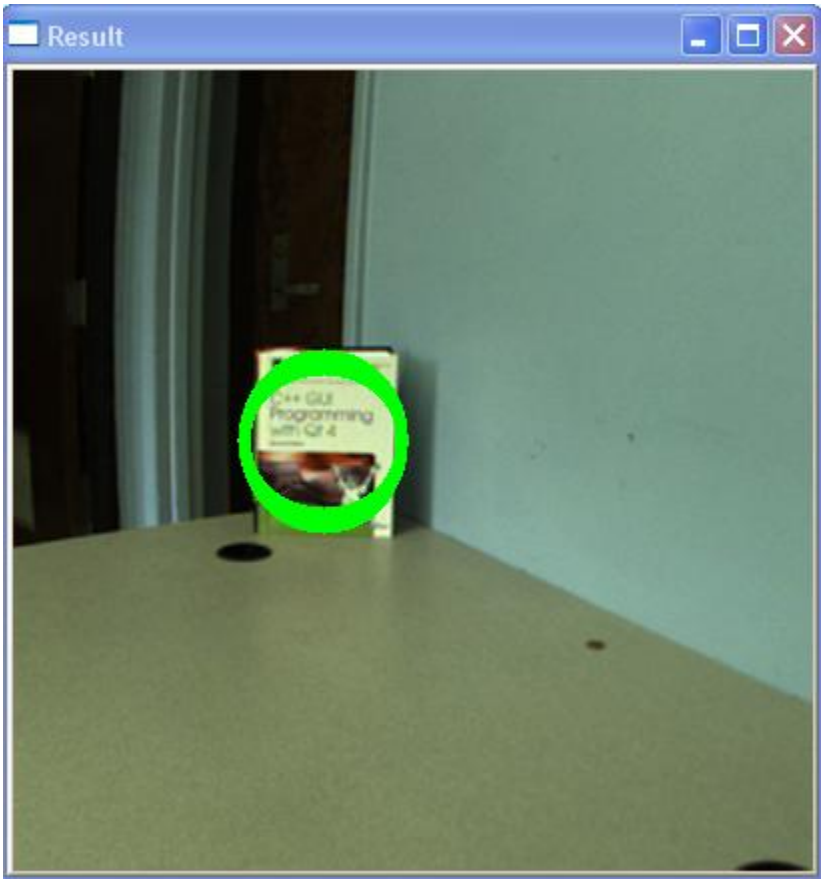

Figure 91 - Result of Template Matching

The second method to be used in comparison with the new Invariant Feature Selection method is artificial neural networks. OpenCV has built-in functionality which is used to train a neural network classifier. This approach was developed by Viola and Jones [39] and later was extended by Lienhart [40,41]. This approach by Viola, Jones and Lienhart is based on Haar Like features as shown in **figure 92.** Haarlike features consist of a set of primitives that can be cascaded

together in a neural network to form a somewhat robust classifier. The specific detail of how this classifier builds this cascade is outside the scope of the dissertation. For further information, one can read about *Face Detection* in the *Intel Technology Journal* [25].

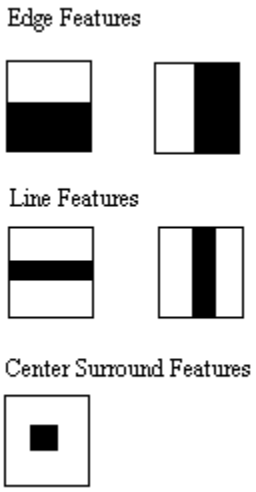

Figure 92 - Haar Like Features

As stated previously, one of the most annoying downfalls of a neural network approach is the massive amount of training that must be used for training. The typical amount of training data consists of at least 1000 positive training images, and at least 4000 negative images: positive images contain the object of interest and negative images do not. Moreover, one has no idea how a classifier will perform after the training of the neural network has been completed.

TESTING

Now that all three methods have been described, they will be compared against each other. For this test, two objects will be selected for tracking. These objects consist of textbooks for indoor conditions and automobiles for outdoor conditions. The QT4 textbook in **figure 91** will be used for indoor testing. For outdoor testing, images of automobiles will be used for testing. The results of testing will be documented in the previously mentioned confusion matrix.

The reason only two objects were used for testing in this dissertation is the same reason that originally motivated this research. **Figure 93** and **figure 94** show the amount of time invested in training each classifier. As one can see in **figure 93**, training the textbook classifier for both the template matching and the invariant feature selector took less than one minute. However, training the neural network classifier for the textbook took 14 hours to complete. Even worse, each positive training image must be manually entered into the neural network training program. This mind-numbing step of manually entering 1000 images took around 4 hours to complete.

Developing the neural network classifier for automobiles was even more tedious. Because of the sheer number of images that needed to be entered into the program, 4 students assisted with the training of the classifier. Once the painstaking task of entering all the images was completed, the neural network classifier took more than 3 days to complete!

	TEXTBOOK TRACKING		
	TEMPLATE MATCHING	ARTIFICIAL NEURAL NETWORKS	INVARIANT FEATURE SELECTION
Number Positive Images	1	1000	4
Number Negative Images	0	4000	0
Training Time	< 1 minute	14 hours	< 1 minute

Figure 93 - Time Spent Training the Textbook Classifier

	FRONTAL AUTOMOBILE TRACKING		
	TEMPLATE MATCHING	ARTIFICIAL NEURAL NETWORKS	INVARIANT FEATURE SELECTION
Number Positive Images	1	5000	2
Number Negative Images	0	10000	0
Training Time	< 1 minute	Approx. 3 Days	< 1 minute

Figure 94 - Time Spent Training the Frontal Automobile Classifier

Now that the training of each classifier is completed, their results will now be compared. Each classifier will be given 100 images (50 Positive, 50 Negative) and the results will be documented. As stated in the *Methodology* section, all indoor testing will be conducted in constant lighting conditions while all outdoor testing will be conducted under clear skies. First, the template matching algorithm will be tested against the QT4 textbook and automobiles. Afterwards, the neural network and invariant feature selection method will be compared against the same objects.

The first method to be tested is template matching. One of the best properties of template matching is the very small amount of false alarms. As shown in **equation 44**, the template matching algorithm simple slides the template image across the full size image. This property allows for near perfect object detection if the full size image contains a near copy of the template image. Unfortunately, if the object in the image is undergoes any form of transformation, the value of **equation 44** grows very rapidly. Because the matching of a template matching algorithm is very specific, this type of object detection is used very heavily in manufacturing processes where objects are kept in very controlled conditions.

Because template matching doesn't work for any scenarios other than an exact match of the object, documenting the results in a confusion matrix wouldn't yield any extra information. Template matching simply provides very rapid object detection under very controlled conditions. Because one of the goals of this research was to dramatically reduce the burden on the engineer in the object detection process, the ease of use for template matching will be used for ergonomic comparison.

The neural network classifier will now be used to attempt to track the QT4 book. The result of the neural network classifier tracking the QT4 book is shown in the confusion matrix of **figure 95**.

As one can see, the neural network classifier completely failed to find the QT4 textbook. Despite being trained with 1000 positive images and 4000 negative images, the neural network classifier missed every single positive image of the QT4 textbook. Moreover, the neural network classifier for the textbook was plagued with false alarms. **Figure 95** shows the results of 100 images, where the classifier was given 50 images containing the QT4 textbook and 50 images that did not contain the QT4 textbook.

For each positive test image, a" hit" was defined if the classifier detected the correct object (the QT4 textbook). If the classifier detected the wrong object in a positive test image, it WAS NOT classified as a false alarm, but rather a "miss". If the classifier detected an erroneous object when given a negative test image, a "false alarm" was declared. When the classifier didn't detect anything in a negative image, a "correct rejection" was declared. This approach was chosen so that the values in the confusion matrix would sum to 100. This approach was kept consistent throughout testing.

Neural Network Textbook Tracking Confusion Matrix	Actual Positive	Actual Negative
Predicted Positive	0	16
Predicted Negative	50	34

Figure 95 - Confusion Matrix for Neural Network Tracking QT4 Book

The reason why the neural network classifier failed to track the textbook is not fully understood. Perhaps the textbook did not contain enough Haar like features to build an accurate classifier. Perhaps Haar like features was simply a poor feature choice. Maybe the classifier simply wasn't given enough training images. Either way, more than 4 hours of manually inserting positive images and 14 hours of classifier training yielded very disappointing results.

The neural classifier yielded much better results when trained to track the frontal view of automobiles. An example of this tracking can be shown in **figure 96**.

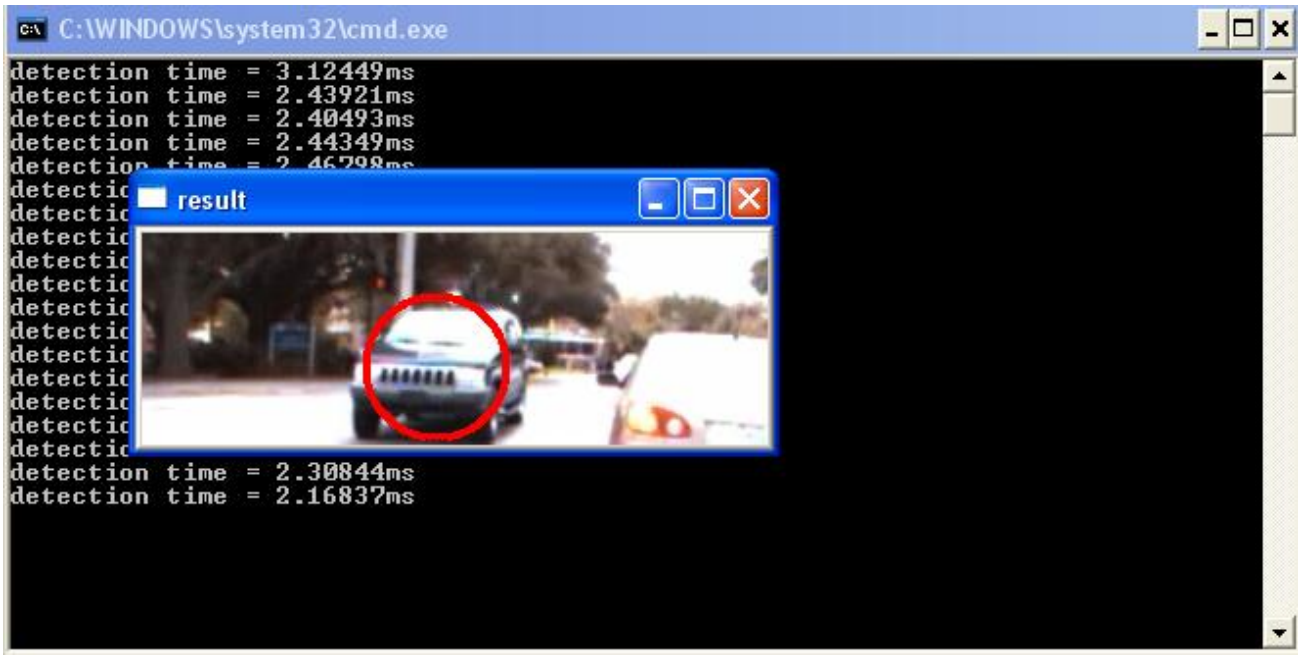

Figure 96 - Neural Network Classifier Tracking Automobiles

The neural network classifier for tracking the front views of automobiles was designed during the 2007 Darpa Urban challenge; a robotics competition where autonomous vehicles were to navigate themselves throughout cities. Because the tracking of automobiles was of substantial importance to

the objective, several people were able to help with the monumental task of training the classifier. The results of the automobile classifier are shown in **figure 97**.

The neural network classifier was trained with massive amounts of automobiles. These automobiles consist of cars, trucks, and vans. The confusion matrix in **figure 97** shows that the neural network classifier was able to track a variety of automobiles 34 percent (17/50) of the time. While this percentage may seem low, one must consider that the raw data coming from a computer vision program is almost never used without some sort of filtering.

Neural Network Automobile Tracking Confusion Matrix	Actual Positive	Actual Negative
Predicted Positive	17	7
Predicted Negative	33	43

Figure 97 - Confusion Matrix for Neural Network Tracking Multiple Automobiles

The Invariant Feature Selection method was then put through the same tests as the previous classifiers. This method has already been shown tracking the QT4 book shown in **figure 86**. This method was able to find the textbook through a series of transformations in space. The confusion matrix for this test is shown in **figure 98**.

Invariant Feature Selection Tracking Textbook	Actual Positive	Actual Negative
Predicted Positive	48	3
Predicted Negative	2	47

Figure 98 - Confusion Matrix for Invariant Feature Selection Tracking QT4 Book

When tracking the QT4 textbook, the invariant feature selection method substantially outperformed the neural network classifier (which wasn't difficult since the neural network classifier didn't detect anything). Unfortunately, the same cannot be said about the automobile scenario. When the goal was to track multiple instances of automobiles, the neural network method was the slight winner. The new invariant feature selection method had a difficult time tracking multiple automobiles with a single classifier. The new method was superior when the automobiles were similar in appearance; however the neural network method pulled slightly ahead when the automobiles were drastically different in appearance. The results for invariant feature selection method tracking multiple automobiles are shown in **figure 99**.

Invariant Feature Selection Automobiles	Actual Positive	Actual Negative
Predicted Positive	14	6
Predicted Negative	36	44

Figure 99 - Confusion Matrix for Invariant Feature Selection Tracking Automobiles

This chapter has illustrated that the invariant feature selection method has an ease of implementation which rivals that of template matching. However, the new method is far more robust than template matching when the object to be detected undergoes spatial transformations. By comparing the new method to template matching, the new method is clearly an improvement.

The new method was then compared to a neural network approach. The new method was vastly superior to the neural network approach when tracking the QT4 textbook. Both the new method and the neural network approach were then given one of the most challenging object detection problems, automobile detection. In this scenario, the neural network approach only slightly outperformed the new method with regards to "hits", despite taking more than 3 days to train the classifier. However, the new method had a slightly lower "false alarm" rate than the neural network classifier. In short, this new method gives the user the ease of template matching with the robustness of a neural network.

CHAPTER 9 - CONCLUSION / FUTURE WORK

This dissertation has shown the development and implementation of a new method to perform object detection in images. With this conclusion, the creation of the individual feature classes and their overall implementation into the program has been illustrated. This new method was then compared to two previously well known methods of performing object detection in images (template matching and neural networks).

While there were several miniature milestones of this research, the overall goal of this research was to place a computer "in the loop" to aid in the feature selection process. The user simply inputs the images he/she wants to track in future images and the program then informs the user which features are best for finding that particular object.

With the computer informing the user as to which features are useful, substantial time is saved. A good example of the amount of time that can be saved was the tracking of the QT4 book using the neural network. One thousand images of the book were entered and trained overnight, only to find that either Haar like features are a poor choice or the user didn't use enough training images. This type of training can be extremely frustrating; especially when the user is on a tight schedule such a finishing a dissertation before a deadline. With this new method the user is relieved of the burden of manually entering thousands of images only to find that his/her feature selection was a poor choice.

This approach of object detection was inspired by the human vision system. Whereas most pattern recognition techniques focus on separating an object from a background, this new method

completely ignores the background. By focusing on the object to be detected instead of the background, the amount of training data is substantially reduced.

The human vision system is holistic, meaning that humans see objects as whole instead of a combination of features. Humans segment (group) different parts of images based on the previously described Gestalt principles and then proceed to classify the individual image segments. By utilizing the growing search window, the holistic natures of human grouping can be mimicked.

This new approach gives the user the ease of implementation of a template matching algorithm with a robustness rivaling that of a neural network classifier. With this new method, determining whether a feature is good for tracking can be performed in seconds, as opposed to days. This topic leads to the future work.

FUTURE WORK

There are three major areas for expansion of this research. Obviously, the first area of expansion is the addition of more feature classes. There are currently twelve feature classes being used for object detection. These twelve features appear to do a reasonable good job of tracking objects. There are many other features that are not included into the program such as Haar like features. If Haar like features were built into the program, perhaps the amount of time training the neural network to find the QT4 textbook could have been avoided. Other possible features include methods such as those used in optical character recognition.

The second way to improve this program is rewriting the search window (**figure 51**) to better utilize multi-core processors. The modern computer industry is moving away from making processors faster (clock speed), and instead are taking the path of adding more processing cores to a single chip. In the year 2000, most people have never heard of a dual core processor for personal computers. Only a few years ago, dual core processors were the standard. Currently, most desktop computers being purchased are equipped with quad-core processors. This trend is expected to continue for the near future.

Programs utilize multi-core processors by creating execution units called *threads*. Threads are lightweight processes that can run simultaneously. If a computer has a multi-core processor and different parts of a program can run in parallel, the user can put these different parts of the program into threads to cut computation time. Despite their usefulness, multi-threaded programs are very difficult to debug. Because the programmer must keep track of several operations occurring in parallel instead of sequentially, threads are a very rich source of unexpected program crashes. Nevertheless, the computing industry has invested substantial time and money into designing multi-core processors, thus forcing programmers to adhere to their philosophy.

The current implementation of the search window is a multi-threaded approach based on the size of the scanning rectangle (**see figure 100**). While this approach satisfies current requirements, it doesn't leave much room for growth. As computers inevitably gain more processing cores, adding more threads in the current manner will only give the search window more resolution: this approach won't necessarily make the program run any faster.

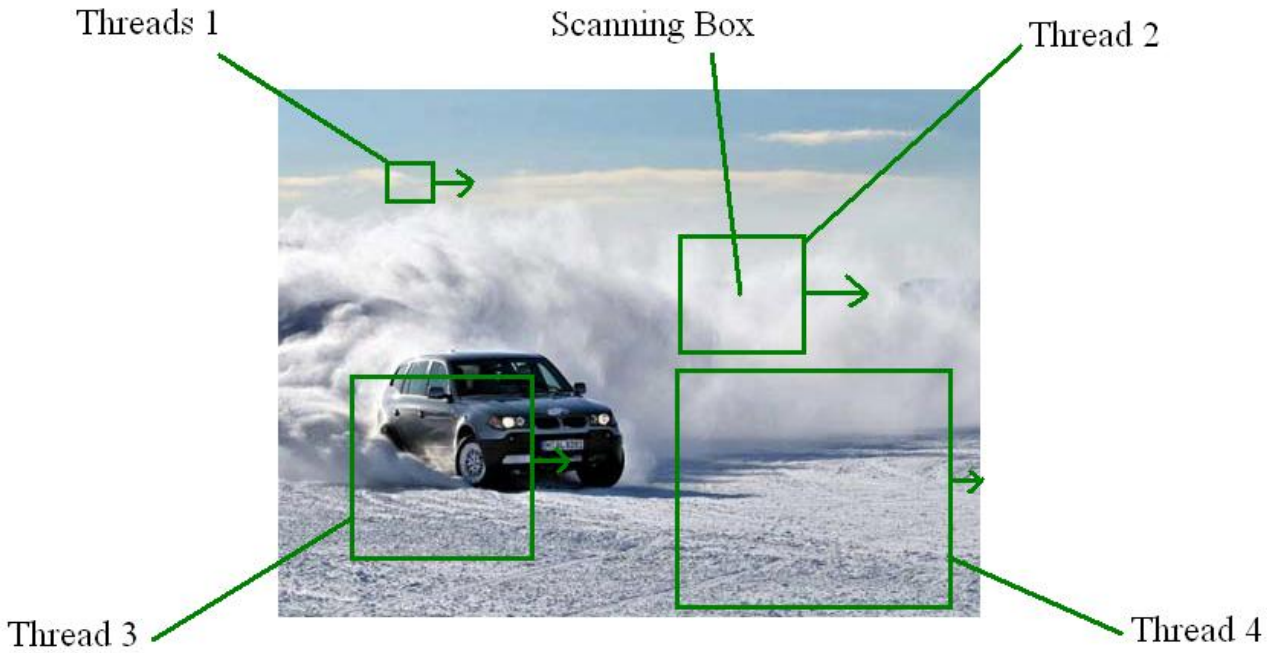

Figure 100 – Current Multi-Threaded Search Window

A much better approach for creating the multi-thread search window is shown in **figure 101**. Instead of basing the threads on the size of the scanning rectangle, the threads are based on the number of processor cores. For example, if a computer has 4 processor cores, the image will be split into four sections. If a computer has 16 processor cores (such as the upcoming AMD "Bulldozer" processor), the image will be split into 16 sections as shown in **figure 102**. Writing the search window in this manner maximizes the future advances in multi-core technology.

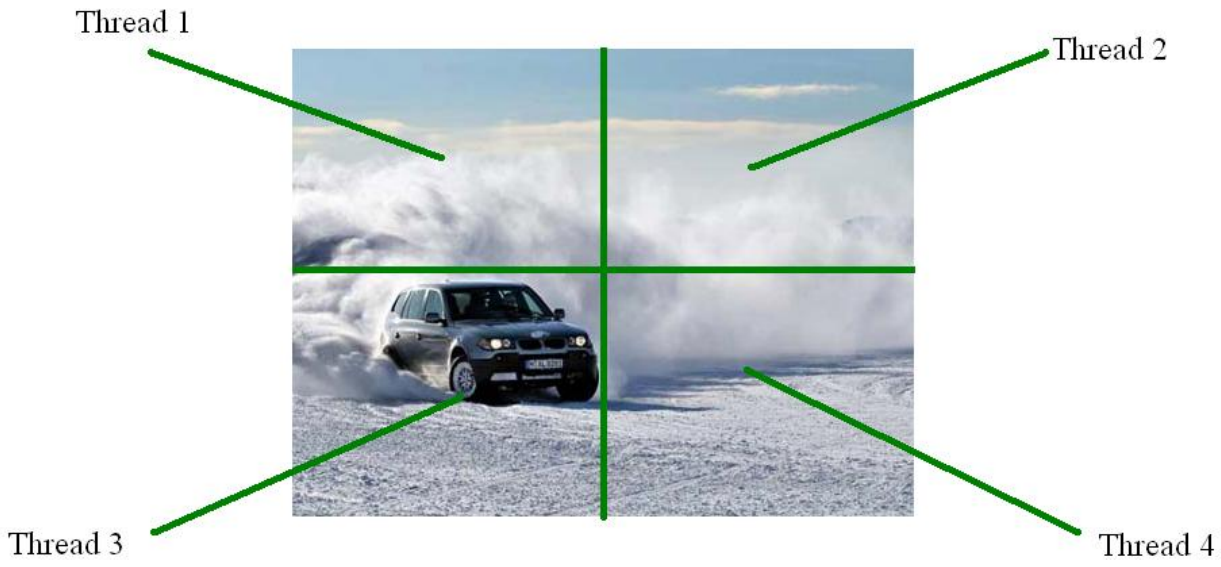

Figure 101 - Future Multi-Threaded Search Window

Figure 102 - Future Implementation of Threading using AMD's Upcoming 16 Core Processor

The third way this program can be expanded is by adding a real-time database of objects. The current model only searches for one object in the image. However, there is no reason the program cannot do multiple object detection. Each object contains features classes which store some parameters about the said object. Some feature classes contain histogram data, others contain simply one value. Each search box is simply compared to the database of objects. If the search window is within the distance threshold of stored object, then the stored object declares a "hit". Expanding the program in this manner would greatly increase its capabilities. For example, not only would the program be able to identify automobiles, the program would be able to tell the user what kind of automobile is being tracked.

Adding more feature classes, improving the search window, and building a database are three ways to allow for future expandability. As more research is performed in the area of human vision, new, more robust feature classes are destined to be created. Old and new feature classes in conjunction with faster processors and a growing database of objects will create an object detection scheme that has a chance of eventually rivaling the human monocular object detection process.

REFERENCES

1. Garbage In, Garbage Out. http://en.wikipedia.org/wiki/Garbage_in,_garbage_out

2. Rafferty, T., May 9, 2004, Goodbye, Evil Robot; Hello, Kind Android, in *The New York Times*.

3. Moran, M., July 25, 2007, The 50 Best Robot Movies, in *Time Online*.

4. Leven, I., 1975, The Stepford Wives.

5. Verhoeven, P., Robocop 1987.

6. Moore, G., 1965, Cramming More Components Onto Integrated Circuits, in *Electronics Magazine*.

7. Frei, M., April 18, 2005, Moore's Law On Chips Marks 40th, in *BBC News*.

8. Hare, B., January 4, 2005, Babies Recognize Face Structure Before Body Structure, in *Blackwell Publishing*.

9. Clark, R., 2007, Resolution of the Human Eye, in *Clarkvision Photography*.

10. Blackwell, J., 1946, Optical Society America, Volume 36, pp. 624-643.

11. Brown, G., How Autofocus Cameras Work.

12. Bianco, C. , How Vision Works.

13. Sternberg, R., 2006, Cognitive Psychology, Harcourt Brace Publishing.

14. Palmer, S. E., 2003, Visual Perception of Objects, in *Experimental Psychology*. Volume 4, pp. 179-211.

15. Perception, MSN Encarta, http://encarta.msn.com/encyclopedia_761571997_2/Perception_(psychology).html

16. Marcum, J., April 1960, A statistical theory of target detection by pulsed radar, in IEEE Transactions on Information. Theory.

17. Green, D.M., Swets J.A, 1966, Signal Detection Theory and Psychophysics, Wiley Publishing.

18. R. Gonzalez and R. Woods., 1992, Digital Image Processing, Addison-Wesley Publishing.

19. I. K. Sethi and R. Jain., January 1987, Finding trajectories of feature points in a monocular image sequence, in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 9 , Issue 1.

20. V. Salari and I. K. Sethi, January 1990, Feature Point Correspondence in the Presence of Occlusion, in *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Volume 12, Issue 1.

21. Acharya and Ray, 2005, Image Processing: Principles and Applications, Wiley-Interscience.

22. Ignizio, J., 1991, Introduction to Expert Systems.

23. Bishop, C., 2006 Pattern Recognition and Machine Learning, Springer Science + Business Media, LLC.

24. Bishop, C., 1995, Neural Networks for Pattern Recognition, Oxford University Press.

25. Chao, L., May 19, 2005, Face Detection, in *Intel Technology Journal*, Volume 09, Issue 01.

26. Seo, N., 2007, Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features).

27. Duda, Hart and Stork., 2001, Pattern Classification, Wiley-Interscience.

28. Richeldi and Lanzi, 1996, ADHOC: a tool for performing effective feature selection, in *Proceedings Eighth IEEE International Conference*, pp. 102 – 105.

29. Zongker and Jain, 1996, Algorithms for Feature Selection: An Evaluation. Pattern Recognition, in *Proceedings of the 13th International Conference*, Volume 2, pp.18 – 22.

30. Lanzi, 1997, Fast Feature Selection with Genetic Algorithms: a Filter Approach, in *Evolutionary Computation, IEEE International Conference*, pp. 537-540.

31. Liu, Dougherty, Dy, Torkkola, Tuv, Peng, Ding, Long, Berens, Parsons, Zhao, Yu, Forman, 2005, Evolving Feature Selection, in *IEEE Transactions on Intelligent Systems*, Volume 20, Issue 6, pp. 64 – 76.

32. Peng, Long, Ding, 2005, Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy, in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 27, Issue 8, pp. 1226 – 1238.

33. Smith, 1978, Color Gamut Transform Pairs, *Computer Graphics*, Volume 12.

34. Murray and VanRyper, 1994, Encyclopedia of Graphical File Formats, O Reilly and Associates.

35. Shapiro and Stockman, 2001, Computer Vision, Prentice-Hall.

36. Fisher, Perkins, Walker, Wolfart, Image Analysis – Connected Component Labeling, http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm

37. Douglas, Pecker,1973, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, The Canadian Cartographer, Volume 10, pp. 112-122.

38. Kleper, Solter, 2005, Professional C++, Wiley Publishing.

39. Viola and Jones, 2001, Rapid Object Detection Using a Boosted Cascade of Simple Features, IEEE CVPR.

40. Lienhart, Maydt, 2002, An Extended Set of Haar-like Features for Rapid Object Detection, ICIP.

41. Kuranov, Lienhar, and Pisarevsky, July 2002, An Empirical Analysis of Boosting Algorithms for Rapid Objects With and Extended Set of Haar-like Features, Intel Technical Report, MRL-TR.