12-26-2014

Cloud-Induced Uncertainty for Visual Navigation

Alyssa N. Gutierrez

CLOUD-INDUCED UNCERTAINTY FOR VISUAL NAVIGATION

THESIS

Alyssa N. Gutierrez

AFIT-ENY-MS-14-D-43

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government.

AFIT-ENY-MS-14-D-43

CLOUD-INDUCED UNCERTAINTY FOR VISUAL NAVIGATION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Alyssa N. Gutierrez, B.S. Engineering Science

December 2014

AFIT-ENY-MS-14-D-43

CLOUD-INDUCED UNCERTAINTY FOR VISUAL NAVIGATION

Alyssa N. Gutierrez, B.S. Engineering Science

Approved:

//signed//	19 Nov 2014
Alan L. Jennings, PhD Chairman	Date
//signed//	19 Nov 2014
Clark N. Taylor, PhD Member	Date
//signed//	19 Nov 2014
Col Matthew D. Sambora Member	Date

Abstract

Visual navigation allows aircraft to operate without reliance on the Global Positioning System (GPS), whose signals can become jammed or blocked. However, its dependence on a clear field of view limits its use. Visual navigation tracks features on the ground in order to calculate an aircraft's position. When clouds are present in the field of view, features can become either fully or partially obscured by the clouds. This occlusion distorts the representation of the features, possibly shifting them within an image. Both the distortion and displacement of features results in inaccuracies in the position solution returned by the visual navigation system.

The purpose of this research is to address the numerical distortion of features due to the presence of clouds in an image. The research aims to quantify the probability of a mismatch between two features in a single image. Finding the probability of mismatching features will describe the likelihood that a visual navigation system incorrectly tracks a feature throughout an image sequence, leading to miscalculations in position.

The research method approaches the problem by first developing an algorithm for calculating the transparency of clouds in images at the pixel level. The algorithm determines transparency based on the color of each pixel compared to the average color of the clouds. The algorithm provides a computationally efficient method for modeling clouds compared to existing physics-based models. The algorithm is then used on a sequence of motion-simulating images to create a dataset of cloudy aerial images. Matching features are then detected between the original and cloudy images, which allows a direct comparison between features with and without clouds. The calculated transparency values are used to segment the detected features into three

categories, based on whether the features are located in the clear regions (no clouds), edge regions (semi-transparent clouds), or cloud regions (opaque clouds) of an image. The error between features on the cloudy and cloud-free images is determined, and used as a basis for generating a synthetic dataset with statistically similar properties. Lastly, Monte Carlo techniques are used to find the probability of mismatching. The method for deriving probability of mismatching features can be used for automatic testing of a visual navigation system to determine in which situations the system is likely fail, versus when it can be expected to succeed. Awareness of these situations will help to promote confidence in visual navigation techniques.

Table of Contents

Page

List of Figures

List of Tables

List of Symbols

List of Abbreviations

Abbreviation		Page
GPS	Global-Positioning System	1
INS	Inertial Navigation System	1
UAV	Unmanned Aerial Vehicle	1
SIFT	Scale-Invariant Feature Transform	12
RANSAC	Random Sample Consensus	20
EKF	Extended Kalman Filter	22
ASIFT	Affine Scale-Invariant Feature Transform	24
PDF	Probability Density Function	25
PMF	Probability Mass Function	25
CDF	Cumulative Distribution Function	25
CTCA	Cloud Transparency Calculation Algorithm	30

CLOUD-INDUCED UNCERTAINTY FOR VISUAL NAVIGATION

I. Introduction

Navigation plays an essential role in many military and civilian missions. With applications such as mapping, surveillance, and transportation, its importance is undeniable. The most commonly-used method for navigation at present is navigation using the Global Positioning System (GPS). GPS-based navigation has proven to be accurate and widely accessible from all reaches of the globe. When paired with an Inertial Navigation System (INS), GPS-based navigation provides information with even higher accuracy. These advantages make GPS-based navigation particularly suitable for unmanned aerial vehicles (UAVs), which rely exclusively on the received information to navigate.

However, GPS devices suffer from known vulnerabilities, such as navigation in GPS-denied environments. GPS-denied environments can occur due to interference, jamming, or signal blockage in urban canyons. In GPS-denied environments, a GPS/INS navigation system is forced to rely solely upon INS. However, the accuracy of INS degrades cumulatively over time without the help of GPS. Thus, an emerging alternative to standard navigation is automated visual navigation (referred to hereafter as visual navigation). Visual navigation tracks features identified in the field of view below the UAV obtained from images from an onboard camera. The use of visual navigation has been increasing as real-time implementation becomes a reality. UAVs are currently not well-equipped to navigate without GPS in cloudy environments, however, due to ground obstruction from the clouds. Thus, the aim of this research is to study the effects of clouds on the tracked features to help address that

vulnerability.

This chapter is organized as follows. First, the statement of the problem to be researched is given in Section 1.1. An overview of the research method is provided in Section 1.2. Necessary assumptions are explained in Section 1.3, followed by a description of the contributions of the research in Section 1.4. Lastly, Section 1.5 provides an outline of the remainder of the thesis report.

1.1 Statement of Problem

As previously mentioned, the use of visual navigation methods has increased with the advance of computational speeds, allowing visual navigation to now be implemented in real-time. This is of great importance to navigation for UAVs, providing them with another tool for reliable performance. Arguably the most important aspect of UAV navigation is increased autonomy for a wide range of flight scenarios. Thus, this research aims to investigate one area of potential growth in autonomy for UAVs: visual navigation in a cloudy environment.

More specifically, this research strives to quantify the uncertainty in visual navigation due to clouds. Clouds present problems to visual navigation systems by affecting the placement and numerical description of visual features. Feature placement is affected because clouds introduce new objects to the image that a feature detector may identify as a distinguishable region. Feature description is affected because the descriptors are based on the gradients in a local area, which are altered when clouds are introduced based on the texture of the clouds.

This research addresses the latter problem: changes in feature description due to clouds. By measuring the error between images with and without clouds, the amount of perturbation in features caused by the clouds can be determined. This will allow the likelihood of mismatching two different features on the same image to be determined.

Finding the likelihood of mismatching is significant because it describes how likely a visual navigation system is to track multiple features throughout an image sequence, believing they are the same feature. This occurrence would cause the visual navigation system to produce incorrect position information over time.

Research is needed in this area because UAVs in GPS-denied environments are vulnerable to miscalculations in position. Thus, higher accuracy in alternative methods of navigation such as visual navigation is necessary. Position inaccuracy has the potential to veer a UAV off course of its trajectory or target. Determining the likelihood of encountering position errors based on the clouds will allow UAVs to know how reliable their calculations are, thus making them more safe, reliable, and less prone to mishandling expensive resources.

1.2 Research Overview

The method derived in this research for determining the probability of a mismatch between features is broken into three main stages. First, an algorithm for calculating cloud transparency is developed. The algorithm extracts clouds from images taken against a blue sky background. The transparency of each pixel in the image is primarily determined by calculating the distance from each pixel's color to the mean color of the clouds. The final output is a template that gives the transparency value of the image for each pixel.

Second, an image database is created for feature detection. Using the algorithm, a database of cloudy aerial images is created by overlaying the cloud templates onto aerial images. Features are detected in the original aerial images, representing the true placement of features, and then on features in the cloudy aerial images using the same feature locations. These features are then categorized according to their placement on the cloudy images: either in regions without clouds, along cloud edges, or fully

within clouds. The categorization is implemented based on the average transparency of the region over which each feature is calculated.

The final stage of the research then develops the method for determining the likelihood of a mismatch. First, a model for describing the error between the original and cloudy features is composed and implemented on each feature pair. A synthetic dataset is created based on the modeled errors. The synthetic dataset is created to reduce dependence on the fixed, raw dataset and to create an arbitrary number of new, statistically-similar observations within the dataset. Lastly, Monte-Carlo simulations are employed using the synthetic dataset in order to find the likelihood that two features within the same image are mismatched. The simulations are run based on each feature region, thus giving three separate analyses for the likelihood of mismatching based on the prevalence of clouds surrounding each feature.

1.3 Assumptions & Limitations

Certain assumptions must be made in order to complete the research. The assumptions and the limitations they introduce are given in the following subsections, organized by the segment of research to which they apply.

1.3.1 Cloud Transparency Templates.

An assumption made during the cloud transparency calculation algorithm of Section 3.1 is that the appearance of clouds from overhead and from below are the same. The cloud template algorithm completed for the first stage of research accepts cloud images as inputs. These images are taken from directly below the clouds against a blue sky background because aerial images taken from overhead would be cluttered and difficult to separate the clouds from the background. This assumption notes that cloud pictures taken from ground view have a similar texture and transparency as

those taken from an aerial view, but do not capture effects such as reflections on the surface of the cloud or shadows produced by clouds. However, the main advantage of having the sky in the background is that the background can easily be estimated to determine transparency.

Additionally, it is assumed that the image input to the cloud transparency algorithm is similar in color to the image used to train the algorithm. The image used to train the algorithm creates a model based on the coloring of the image, and thus can only be accurately used if the input image has similar color values.

1.3.2 Cloud-Induced Feature Error.

Although there are many branches of research into visual navigation, only the effects of clouds will be studied in this thesis. Other current research areas in visual navigation include handling delayed or faulty observations, improving consistency, and reducing computational requirements [1]. Additionally, this research addresses the effects of clouds for a navigation system that has access to a camera only. Although UAVs commonly pair a visual system with an INS, for this research, it is assumed that only the camera will be used as a navigational aid.

As mentioned in Section 1.1, when clouds are introduced into an image, both the feature placement and feature description are altered. However, since this research focuses on looking at the differences in a particular feature when clouds are added to it, it must be known that the cloudy feature matches to the original feature. Thus, the scope of the research prevents investigation into the change in feature placement due to clouds. For this reason, features are forced to be in the same arbitrary locations, even though this would not occur in a real-life scenario.

1.3.3 Synthetic Dataset Creation.

When developing the method for the synthetic dataset creation, principal component analysis is used to reduce the dimensionality of the data. When reducing the number of dimensions from the feature descriptors, it is assumed that the selected number of retained dimensions is sufficient.

Additionally, after creating the dataset, it must be assumed that the synthetic dataset accurately represents the data on which it was based. An analysis is performed on the synthetic dataset, but due to its qualitative nature, does not provide a numerical threshold for which the data can be considered sufficient or not.

1.4 Research Contributions

The research develops an algorithm for calculating the transparency of clouds within images at the pixel level. The result is a method that can overlay clouds with various structures on top of any desired image to produce realistic cloudy scenes. The developed algorithm can be adapted for applications beyond the use of this research, such as for detecting other objects that have cloud-shaped structures.

The primary contribution of this research, however, is to investigate and quantify the errors in features due to clouds. The research will compare features in two different scenes: a scene looking down at the ground, and the same scene with clouds introduced. The uncertainty to be measured is the likelihood that a particular feature within an image will be mismatched to another feature in the same image. This determination will give insight into how likely a visual navigation system would incorrectly identify a feature in an image, leading to miscalculations in position.

1.5 Outline

Subsequent chapters of this thesis are organized as follows. Chapter II provides an in-depth review of the background required to understand the context and methodology for the research. Chapter III describes the methods with which the necessary data will be collected, and further, how the data will be evaluated in order to solve the research problem. Chapter IV details the collected data and provides a thorough analysis of the results found. Lastly, Chapter V summarizes and concludes the research process, as well as provides suggestions for future work on the research problem.

II. Background

This chapter presents an overview of the background necessary for the research as well as an overview key methods to be used for the research. Section 2.1 describes various methods with which cloud modeling is accomplished, since the research requires adding clouds to aerial images. Section 2.2 gives an overview of visual navigation, including common feature detection algorithms, alternatives to feature-based detection, and how visual navigation updates the position solution during flight. Section 2.3 details common challenges for visual navigation systems and how these challenges have been addressed. Section 2.4 explains essential topics for stochastic modeling. The mathematical notation used for the research is detailed in Section 2.5. Lastly, the chapter is summarized in Section 2.6.

2.1 Cloud Modeling

Modeling realistic clouds has challenged researchers since the 1980s. The poorly-defined surfaces and boundaries of the clouds, along with their varying degrees of transparency, present many problems to standard image generation techniques [2]. The often complex mathematical models produce successful results at the expense of high computational costs.

One of the early attempts of modeling clouds at reduced costs involved creating textured ellipsoids. These ellipsoids used a mathematical texturing function that introduced phase shifts for the appearance of randomness [2]. Although the ellipses could be linked together to generate more variety of clouds types, this method does not emulate the true structure of clouds. An alternative popular modern method of creating synthetic clouds is known as Perlin noise. Perlin noise is capable of creating textures such as clouds, marble, and water that is pseudorandom in appearance. It

creates the textures by generating smoothed noise over an area or volume. Perlin noise is useful when faced with memory or bandwidth limitations, with the tradeoff that it cannot simulate atmospheric illumination effects [3]. Other simulation methods embody more in-depth analysis of cloud structures. These physics-based approaches use equations that model fluid flow, thermodynamics, water condensation, and evaporation to generate clouds [4]. The drawback to this approach is that it relies on inputs such as velocity, air pressure, and temperature. This information may not always be attainable beforehand or at the time of need.

This research avoids completing analysis on unrealistic datasets by avoiding synthetic cloud generation techniques altogether. Instead of using a model to attempt to create structurally-sound clouds, images of real clouds are used. By using these images, all of the naturally present features of clouds are captured. The features are derived from the environment in which the clouds were created, such as transparency, shadows, wispiness, robustness, and more. Computer-generated clouds may impose an unrealistic structure which, in turn, would create unrealistic results. The clouds would be either too structured or too random, not reflecting the effects of nature, and thus may provide different results than could be expected in the field. Thus, high-resolution photographs are used to capture true cloud formations.

2.2 Visual Navigation

The concept of visual navigation is to allow an autonomous vehicle to navigate using an on-board optical sensor, without the aid of GPS. The most prevalent method for visual navigation is feature-based navigation, although several other methods exist, such as optical flow and image mosaicking. This section describes various common feature-detection algorithms, alternatives to feature-based navigation, and how a visual navigation system updates its position as it travels.

2.2.1 Feature Detection Algorithms.

Feature detection is an essential computer vision technique that can be used to generate and describe local features in images. Among the common applications for feature detection are object recognition, robotic mapping, visual navigation, image stitching, and camera calibration. The definition of a "feature" varies according to the application in which it is used; therefore, many different feature detection algorithms exist, each with its own strengths and weaknesses. For the purpose of visual navigation, real-time feature detection is necessary.

A wide variety of feature detection algorithms exist, each of which strive to optimize performance for a given scenario. Generally, a feature detection algorithm consists of two segments: detection and description. The detection process finds specific regions of interest within an image. The description process uses the local areas defined by the detectors to assign numerical values to the regions to create distinctive features. These descriptors are used in the feature matching process and are the subject of this investigation. The following subsections describe several of the most popular detection algorithms.

2.2.1.1 Histogram of Oriented Gradients.

The Histogram of Oriented Gradients (HOG) algorithm was developed in 2005 by Dalal et al [5]. The method of generating HOG descriptors is based on evaluating local normalized histograms of image gradient orientations within a dense grid. The overall idea behind the method is based on the observation that the shape and appearance of an object within a local area can often be well-characterized by the distribution of local intensity gradients.

The HOG algorithm uses a sliding window approach to find distinctive features. Instead of having a stage of the algorithm specifically dedicated towards finding the

regions of interest, the entire image is looked at one window at a time. Each window is divided into small spatial regions known as "cells". A local histogram of gradient directions is then calculated over the pixels within each cell. The histograms are then combined to form the feature descriptor. Lastly, each of the cells are normalized over larger regions called "blocks" to provide improved invariance to illumination changes. These normalized descriptor blocks are the HOG descriptors.

Though very effective for detection of human faces, HOG descriptors are not ideal for visual navigation. HOG descriptors are not scale or rotation invariant and do not allow for feature matching, only feature classification. Thus, HOG descriptors are not suitable for the research.

2.2.1.2 Viola-Jones Detector.

The Viola-Jones algorithm was introduced by Viola and Jones in 2001 [6]. This feature detection algorithm is particularly suited for real-time object detection due to the introduction of integral images. Using integral images greatly reduces computational speeds compared to using raw images for analysis. A more thorough explanation of integral images is given in Section 2.2.1.4. The Viola-Jones algorithm uses three types of features to detect objects: two-rectangle, three-rectangle, and four-rectangle features. A representation of these features is shown in Fig. 1. The features are calculated by subtracting the sum of the pixels that lie within the white rectangles from those in the grey rectangles.

After feature identification, the detector selects a small number of important features and uses them to train efficient classifiers. The image is partitioned into sub-windows before detecting objects. During the learning process, a cascade of weak classifiers is trained. If the sub-window fails to pass any of the classifiers at any stage, it is immediately rejected and the algorithm proceeds to the next sub-window.

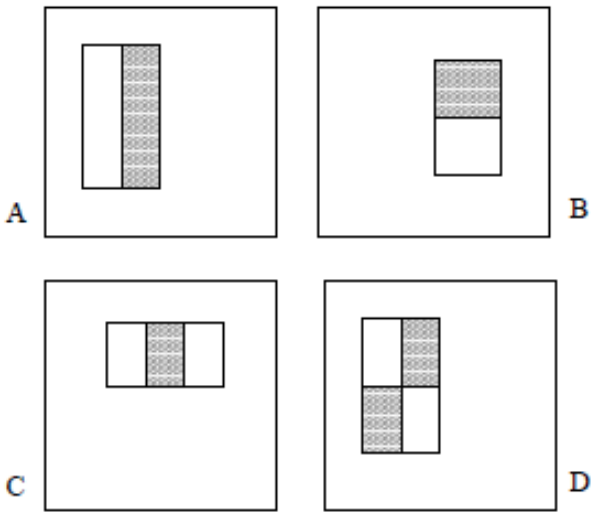


Figure 1. Features used in the Viola-Jones detector algorithm. Two-rectangle features are given by A and B, three-rectangle features are given by C, and four-rectangle features are described by D [6].

Subsequent classifiers are then trained using the successful examples, leading to progressively more complex classifiers.

The Viola-Jones detector is a fast method for identifying features, but is not practical for this research due to the structure of the features. Simple rectangular features are not rotation invariant, making them unsuitable for visual navigation, where the same feature must be robustly detected when seen from various points of view.

2.2.1.3 Scale-Invariant Feature Transform.

Lowe created the Scale-Invariant Feature Transform (SIFT) algorithm in 1999 [7]. SIFT was designed specifically to be invariant to changes in scaling, translation, and rotation, as well as partially invariant to changes in illumination and affine projection [7].

The process of identifying and describing features in SIFT is divided into four parts. The first stage is the scale-space extrema detection, which searches for poten-

tial interest points over all scales of the image using a Difference-of-Gaussian function. Second, the keypoint localization process determines the location and scale of all candidate interest points. Next, each keypoint is assigned an orientation based on local image gradient directions. Lastly, the keypoint descriptor is formed by measuring the local image gradients at the specified scale in the region surrounding the keypoint location.

SIFT is one of the most common algorithms employed in visual navigation due to its invariance properties and its robustness. Features detected using SIFT are highly repeatable, meaning the algorithm can be relied upon to produce the same results over many implementations. SIFT is not used for detecting features in this research, however, because the SURF algorithm maintains comparable results to SIFT, but at reduced computational costs. The SURF algorithm is explained in detail in the following subsection.

2.2.1.4 Speeded-Up Robust Features.

SURF was developed in 2006 by Bay et al. [8] as an improvement to the existing SIFT algorithm. Like its predecessor, SURF is scale- and rotation-invariant, making it ideal for visual navigation purposes. The development of SURF was aimed specifically at increasing the speed of SIFT while maintaining comparable results. The primary difference between SIFT and SURF is SURF's use of integral images, introduced by Viola and Jones. Using integral images for feature extraction instead of the original input image allows SURF to attain computational speeds three to five times higher than those of SIFT [9].

The SURF algorithm, like many feature detection algorithms, is divided into two main tasks. The first task is to find points of interest within the image, also referred to as "feature points." These feature points are selected to make them as distinct

as possible, such as areas with high contrast changes. Secondly, the feature points are described in such a way that makes them distinct, robust, and capable of repeatedly being detected. Lastly, the feature points must be matched among images. Although the SURF algorithm does not explicitly define a unique or optimal matching algorithm, matching is nevertheless usually the main purpose for feature detection. Thus, a simple matching criteria is typically applied after detecting and describing the feature points in order to show the success of the feature detection. The following sections explain the purpose of each of the tasks and the process required to accomplish them.

Preprocessing. Before applying the SURF algorithm to the image, any color image must be converted to greyscale, as SURF and the majority of other feature detection algorithms use only pixel intensity for calculations. Additionally, before beginning the feature extraction process, the integral image for the input is calculated. As previously mentioned, using integral images is the primary factor that allows SURF to compute at faster speeds than SIFT. Integral images are a simplified version of the original inputs, yet still retain all of the numerical information about the image. This method for intermediate representation of images was made popular for computer vision applications by Viola and Jones in 2001 [6]. Integral images are created by considering a rectangular portion of an image with area A and coordinates of L_1, L_2, L_3, and L_4. The sum of the pixel intensities within region D can then be calculated using only four operations [6]:

$$A = L_4 + L_1 - (L_2 + L_3). \tag{1}$$

Figure 2 shows a visual representation of the calculation.

The result of (1) is that the pixels of the integral image are simply the cumulative

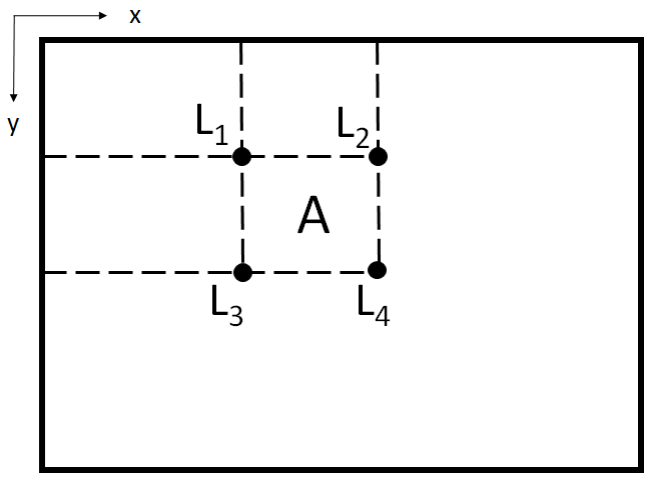


Figure 2. Visual representation of the calculation of integral images. Area A contains the cumulative sum of the original pixel intensities [10].

sum of each of the original pixel values of that area, starting from the top left corner and progressing to the bottom right corner of the image. SURF uses integral images instead of intensity images because the calculation of integral images takes only four operations regardless of the size of the image, thus speeding up the computational time.

Detect Feature Points. After pre-processing the image, the feature point extraction begins. In order to detect interest points in an image, SURF takes a blob-detection approach using the Hessian matrix. More specifically, an interest point is detected where the determinant of the Hessian matrix is a maximum. The Hessian matrix is a matrix representing the second-order partial derivatives of a function. Equation (2) gives the two-dimensional Hessian matrix used in SURF [8]:

$$\mathbf{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}. \tag{2}$$

In (2), $\mathbf{H}(\mathbf{x}, \sigma)$ gives the Hessian matrix at the point $\mathbf{x} = (x, y)$ at a scale of σ. $L_{xx}(\mathbf{x}, \sigma)$ is the "convolution of the Gaussian second order derivative $\frac{\partial^2}{\partial x^2} g(\sigma)$ with

the image **I**," and likewise for L_{xy} and L_{yy} [8]. The derivatives are referred to as the Laplacian of Gaussians.

However, instead of finding the true Laplacian of Gaussians, SURF approximates them as box filters. Figure 3 shows the approximated filters.

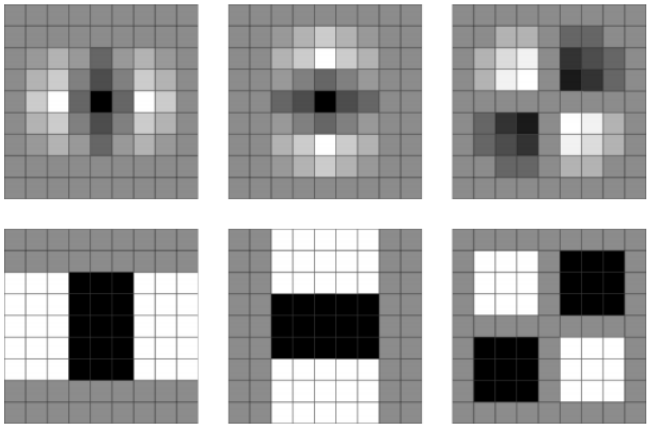


Figure 3. Top row, left to right: Discretized Laplacian of Gaussians (L_{xx}, L_{yy}, and L_{xy}). Bottom row, left to right: Box filter approximations (D_{xx}, D_{yy}, and D_{xy}) [11].

This process results in the approximated Hessian determinant, as given in (3) [8]:

$$det(\mathbf{H}_{approx}) = D_{xx}D_{yy} - (0.9D_{xy}^2). \tag{3}$$

As previously mentioned, the maximum of the determinant of the Hessian matrix is used to detect feature points. After applying (3) to the integral image, the resulting responses are found.

Describing Features. After the set of feature points has been extracted from the image, the next step is to describe each point in such a way that is as invariant as possible to rotation, illumination, and scaling. To achieve this goal, Haar wavelet responses in both the x and y direction are used to determine the dominant orientation of the feature point. These responses give a sense of the direction of the change in pixel intensities within a region. Because they look at the changes in

intensities only, and not the intensities themselves, the Haar wavelet responses remain constant under changes in illumination. The Haar wavelet responses are calculated in a radius of size $6s$ around the detected feature point, where s is the scale at which the feature point is detected. To give a greater importance to the intensity changes closest to the detected feature point, each value within the circular region is weighted with a Gaussian distribution.

Upon calculating the responses in the x and y directions, their sums are taken within a sliding window of size $\frac{\pi}{3}$. The dominant orientation is then determined by the largest vector of sums over all windows. Using the dominant orientation, a square window is created, centered on the interest point and oriented in the direction of the dominant orientation. This descriptor window is then divided into a 4x4 area of subregions. The Haar wavelets in the x and y directions are calculated for 25 points within each of the 16 subregions. For each subregion, the sum of the change in x, change in y, absolute value change in x, and absolute value change in y are calculated and combined to create a 64-dimensional "feature descriptor" for each feature point. These descriptors serve to uniquely identify the feature points. A visual representation of the descriptor-building process is seen in Fig. 4.

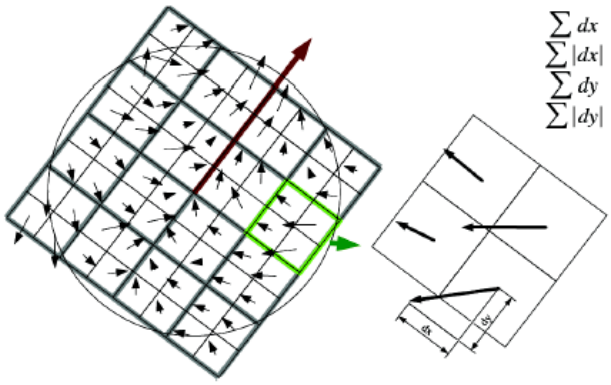


Figure 4. The descriptor window is divided into 4x4 subregions, centered at the feature point and pointing in the dominant orientation. The descriptors are formed by summing the changes in x and y within the subregions [8].

Matching Features. Typically, the next stage of the feature detection process is matching features between images. Many different feature matching algorithms exist. Like the feature detection algorithms themselves, the optimal algorithm largely depends on the application. Some examples of feature matching techniques include k-d trees, sum of squares distance, and Mahalanobis distance. To fit the scope of this research, the matching process is skipped and instead, the features between two images are forced to match. Chapter III describes how the forced matching is accomplished.

2.2.2 Alternatives to Feature-Based Navigation.

Other methods for navigation using an optical sensor exist besides feature-based navigation. In particular, two main alternatives are optical flow and image mosaicking, which are discussed in the following subsections.

2.2.2.1 Optical Flow.

The goal of optical flow is to approximate the 2-D motion field (a projection of the 3-D velocities) using spatiotemporal patterns of image intensity [12]. To do so, optical flow measures the motion of each pixel between image pairs. With this information, a navigating vehicle is capable of detecting obstacles within its path or the trajectories of objects [13]. Optical flow is best suited for scenarios in which the motion between consecutive frames is expected to be small [14]; the accuracy of the solution degrades when subject to large motion differences between frames. For a UAV flying at low altitude, the motion between frames can potentially be large depending on the speed of the aircraft. For this reason, optical flow is not used for this research.

2.2.2.2 Image Mosaics.

Navigating through the use of image mosaics is an emerging technique for visual navigation. Xu and Negahdaripour present a method for real-time estimation of the 3D motion of an autonomous underwater vehicle (AUV) using video images [15]. The method is able to both maintain a specified short-range trajectory and construct an image mosaic of the ocean floor by registering consecutive frames. To maintain the trajectory, camera motion between frames is estimated directly from the variations in brightness in the image sequence. The estimated camera motion is then used to estimate the incremental motion of the vehicle between sequential frames. The motion estimation algorithm is able to estimate 3D translation as well as yaw motion. After the motion has been determined, image registration and removal of distortion is applied to create the image mosaic. Although this method is useful due to its simplicity and its ability to construct a visual map during navigation, the accuracy of the motion estimation is highly sensitive to biased errors, which cause the vehicle to drift from the prescribed trajectory over time. The algorithm also assumes that the motion between consecutive image frames is no more than a couple pixels. While realistic for a slow-moving underwater vehicle, this method is not practical for an airborne vehicle.

2.2.3 Map Registration & Position Updates.

After identifying features within consecutive images, the features must be matched in order to determine the movement of the camera between frames and, consequently, the movement of the vehicle. This is done through image registration, where the transformation matrix that takes the image from one frame to the next allows the images to be aligned correctly. The accuracy of the image registration is directly impacted by the accuracy of the matched features. In order to suppress the likelihood

of incorrect matches between images, the Random Sample Consensus (RANSAC) algorithm is used to both eliminate uncharacteristically poor matches and to determine the transformation matrix between images, known as the image homography. After map registration has been completed, the navigation system can then update the aircraft's position using SLAM. An overview of RANSAC and SLAM is provided in the following subsections.

2.2.3.1 Random Sample Consensus.

The RANSAC algorithm is an important tool for computer vision problems, introduced in 1980 by Fischler et al. RANSAC is an iterative algorithm whose purpose is to find the optimal model for a set of data. The algorithm is intended for use on data with a significant portion of gross outliers. Unlike methods such as least-squares fitting, the iterative method of RANSAC allows outliers in the data to have little effect on the resulting model.

The basic flow of the algorithm is as follows [16]. First, a set of data points P is collected that follows the model M. At least n data points must be collected from P in order to create the model; therefore n must be no greater than the number of points in P. Second, a randomly selected subset S_1 is chosen, consisting of n data points from the set P. The points from S_1 are used to instantiate the model M_1. Third, the model M_1 is used to determine the subset S_1' of points in P that are within a specified error tolerance of M_1. S_1' is referred to as the consensus set of S_1. If the number of points in S_1' is greater than some threshold t, S_1' is used to compute a new model M_1'. However, if the number of points in S_1' is less than t, a new random subset S_2 is selected and the above process is repeated. After k trials, if no consensus set with t or more members has been found, then the process is terminated. The termination ends either in failure or uses the model with the largest consensus set found. A flowchart

representation of the described process is shown in Fig. 5.

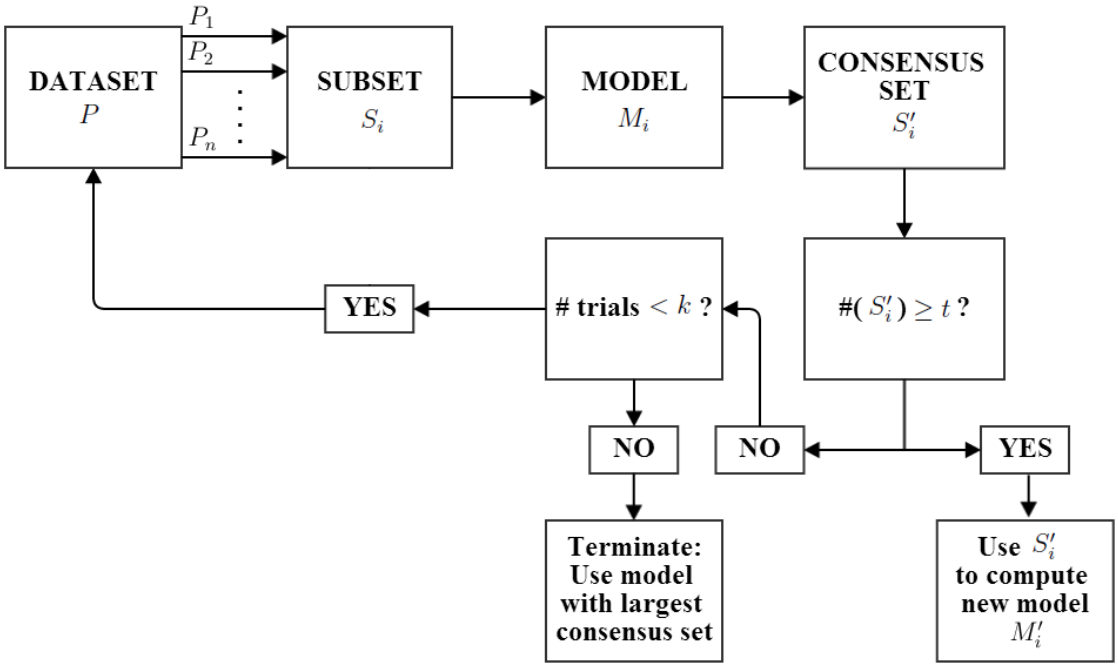


Figure 5. A flowchart representation of the iterative RANSAC algorithm.

In a visual navigation system, RANSAC is employed to find the relationship between two images in terms of the image rotation, scaling, and translation from one image to the next. This relationship can be described by a 3x3 matrix known as the image homography. As explained by Fischler et al., RANSAC is particularly suitable for image analysis because feature detectors often make mistakes [16], meaning that the data to be analyzed can include large errors that would significantly affect a least-squares approach. By using RANSAC, a more accurate approximation of the image homography can be obtained because the outliers in the data are identified and disregarded from the computation.

2.2.3.2 Simultaneous Localization and Mapping.

After image registration between subsequent images, the navigation system can update the position of the vehicle. One way this is achieved is through Simultaneous

Localization and Mapping (SLAM). SLAM allows a navigating vehicle to be placed in an unknown location within an unknown environment and to be able to simultaneously build up a map as well as determine its location within the map. As such, the vehicle's trajectory and the location of identified landmarks are estimated on-line without any *a priori* information about its location [17].

To use SLAM, specific information about the vehicle must be obtained at every time step k [17]:

- $\mathbf{x}_k$: The state vector that describes the vehicle's location and orientation
- $\mathbf{u}_k$: The control vector (applied at time $k-1$) to drive the vehicle to state $\mathbf{x}_k$ at time k.
- $\mathbf{m}_i$: A vector giving the true location of the stationary i^{th} landmark.
- $\mathbf{z}_{ik}$: An observation from the vehicle of the location of the i^{th} landmark at time k.

In addition, the history of all obtained vehicle locations, control inputs, true landmark positions, and landmark observations is maintained. These values are used along with a state propagator, such as the Extended Kalman Filter (EKF). The EKF is used to provide position updates at every time step. The vehicle detects landmarks within the environment at every time step and attempts to associate them with previously identified landmarks. Landmarks that are successfully re-identified are then used to update the vehicle's position in the EKF. Newly-observed landmarks are added to the EKF for future associations and to help to build a more complete map.

2.3 Problems with Vision Systems

Basing a navigation solution on a single sensor will undoubtedly face challenges to overcome, no matter what type of sensor is in use. In the case of a camera, common

problems with visual navigation are obstruction, changes in object view angle, and repetitive patterns within the environment. The following subsections describe these challenges and some of the attempts made to remedy them.

2.3.1 Obstruction.

One key challenge for visual navigation systems is maintaining an accurate position solution when faced with obstruction. To an optical sensor, a partially obscured object has entirely different properties than an object in clear view. Thus, an important task for visual navigation systems is to be able to find and identify partial objects and match them to the same object in other images. This task is partially accomplished through feature detection algorithms such as SIFT and SURF. These algorithms have proven to be able to robustly detect and match features with partial obstruction [7]. Another successful method for object recognition under partial obstruction is the "eigen window" method. This method stores multiple partial appearances of an object in an eigenspace and is able to determine the object as well as its pose [18]. The main drawback to this approach is its vulnerability to changes in illumination and its computational expense.

2.3.2 View Angle Changes.

Another difficulty for visual navigation systems is dealing with changes in the viewing angle of an object. Although SIFT and SURF are designed to be invariant to scale, rotation, translation, and partially to illumination, these algorithms perform poorly when subject to affine distortions. When a camera sees an object whose viewpoint has changed, either due to object motion or change in camera angle, the image gradients change along with the image of the object, affecting the feature descriptors [19]. To improve performance under affine distortion, the Affine Scale-Invariant

Feature Transform (ASIFT) was developed by Morel and Yu. ASIFT, based on the original SIFT, provides full affine invariance by simulating all image views obtainable by varying the latitude and longitude angles along the camera axis orientation [20]. Further research, developed by Jurado, uses predictive affine distortion modeling based on ASIFT [19]. This algorithm improves upon ASIFT by requiring only the generation of a single affine-transformed image, with distortion parameter estimations provided by an EKF.

2.3.3 Repetitive Patterns.

Visually identical features present a difficult obstacle for vision systems. Repetitive patterns, such as tiled floors or a forest of trees seen from above, provide few distinct features with which to match and estimate position. Typically when faced with a repetitive environment, external markers are placed in the scene to aid with navigation. Villamizar et al. present an algorithm that successfully calculates the pose of a UAV without the use of external markers [21]. The algorithm uses a Random Ferns classifier, since feature detection algorithms cannot reliably find distinctive interest points in visually identical environments. The first stage of the algorithm synthetically warps a sample of the target by various shifts and planar rotations, which generates new samples of the target associated with specific viewpoints. The combined collection of samples is used offline to train the classifier. The second stage evaluates the classifier for each input image and its detections are used to update posterior probabilities of different views of the target. This process allows the classifier to adapt to target changes unsupervised.

2.4 Stochastic Modeling

Stochastic modeling is a technique in which a system can be more realistically modeled by incorporating non-deterministic measurements. This allows the modeled system to be determined probabilistically by incorporating known uncertainties. By incorporating random variables into the system modeling, stochastic modeling can help to determine the likelihood of outcomes for a given scenario. The behavior of the random variables is determined by their probability density/mass functions (PDFs/PMFs) or cumulative distribution functions (CDFs). These probability functions can be used within Monte Carlo simulations to determine the probabilistic outcome of the system. An explanation of the probability functions and Monte Carlo simulations is given in the following subsections.

2.4.1 Distribution Functions.

A PDF is associated with a continuous random variable, whereas a PMF relates to a discrete random variable. Both PDFs and PMFs give "point probabilities" of random variables [22]. In the discrete case, the point probability gives the probability $f_X(x)$ that a random variable X is equal to the value x:

$$f_X(x) = P(X = x). \tag{4}$$

In the continuous case, the point probability for X equal to x is always zero, so the definition is altered. Instead, the PDF describes the probability that the random variable falls within the interval from a to b:

$$f_X(x) = P(a \leq X \leq b). \tag{5}$$

Furthermore, PMFs and PDFs also must satisfy the following properties [22]:

$$f_X(x) \geq 0, \ \forall x \in S$$
$$\sum_{x \in S} f_X(x) = 1 \text{ (PMF), or } \int_S f_X(x)dx = 1 \text{ (PDF),}$$

where S is the support of the random variable X.

Another representation of the PDF and PMF is known as the cumulative distribution function. The CDF gives the probability $F_X(x)$ that the random variable X is less than or equal to x:

$$\begin{aligned} F_X(x) &= P(X \leq x) \\ &= \sum_{t \leq x} f_X(t) \text{ (discrete)} \quad (6) \\ &= \int_{-\infty}^{x} f_X(t)dt \text{ (continuous).} \quad (7) \end{aligned}$$

The CDF of a random variable must be non-decreasing and right-continuous. Additionally, $\lim_{x \to -\infty} F_X(x) = 0$ and $\lim_{x \to \infty} F_X(x) = 1$ must be satisfied.

For discrete random variables, PMFs and CDFs share the same information, and likewise for PDFs and CDFs when dealing with continuous random variables. Each function describes the distribution of the random variable in question. These functions are used to determine properties of random variables such as their mean and variance.

2.4.2 Monte Carlo Simulations.

Monte-Carlo methods are those which utilize repeated random sampling to approximate solutions to quantitative problems. The Monte-Carlo method is particularly useful for numerically solving mathematical problems when the analytical solution is either unknown or too complex to determine. The first article detailing the

method was written by Metropolis and Ulam in 1949, though using random numbers to solve statistical problems had been demonstrated well in advance of the article's publication [23].

Monte-Carlo simulations are widely used in mathematics, physical sciences, and engineering in order to approximate the statistics of particular types of data. Collecting data from multiple runs of an experiment is a standard procedure, as more data generally lead to more accurate analysis of the data by the Central Limit Theorem. However, many scenarios allow for only a limited amount of data collection due to time, financial, computational, or any number of other constraints. Due to this restriction, a true distribution of the data can be difficult or impossible to obtain. Thus, Monte-Carlo simulations strive to approximate the unknown distribution of the given data through the use of repeated random sampling.

A typical Monte-Carlo simulation follows three main steps. The first step involves identifying the domain of possible inputs for a particular dataset of interest. Next, random inputs are generated from a probability distribution over the specified domain. Lastly, the individual runs are combined to give the results of the simulation.

A simple example of this process is the determination of the π using Monte-Carlo simulations [24]. Consider a circle inscribed within a square. The value of π can be calculated as four times the ratio of the area of the circle to the area of the square, as seen in (8):

$$4 * \frac{A_{circle}}{A_{square}} = 4\left(\frac{\pi R^2}{4R^2}\right) = 4\left(\frac{\pi}{4}\right) = \pi. \tag{8}$$

Therefore, one simply needs to know the area of the circle and the area of the square to obtain π. To approximate π using Monte-Carlo methods, the first step of the process is to determine the domain over which the simulation will be run. The domain in this example is the entire region of the square. Next, consider dropping grains of sand uniformly over the domain. Using a uniform distribution ensures that there are no

biases to the estimation. Lastly, counting up the number of grains within the circle, dividing by the number of grains in the square, and multiplying by four will give an approximation of π.

Clearly, this approximation is limited by the number of grains of sand available. A small number of grains cannot accurately represent the areas of the two objects, and thus will not give accurate results. However, one can conceptualize that with an infinite amount of sand, the true value of π would be obtained. This approach is not realizable, however, since it is impossible to pour infinitely many grains of sand over a square. Accurate approximations can still be made, though, using finite grains of sand many times over. Due to the nature of random sampling, the first time the sand is poured, the approximation may be too high; the next time, it may be too low. Over the course of many realizations, however, the average of the approximations will converge to π, due to the Central Limit Theorem. Thus, the uncertainty of the results of Monte-Carlo simulations is a function of the number of realizations performed [25], as well as how many samples are within each realization.

2.5 Mathematical Notation

The notational conventions used for the remainder of the document are as follows:

- **Scalars**: Scalars are denoted with either uppercase or lowercase italic letters (e.g., a or A).
- **Vectors**: Vectors are represented by lowercase boldfaced letters (e.g., $\mathbf{a}$). The transpose of a vector is denoted using the superscript $\top$ (e.g., $\mathbf{a}^\top$). All vectors are assumed to be column vectors unless accompanied with the superscript $\top$. The i^{th} instance of a vector is given with a subscripted lowercase letter (e.g., $\mathbf{a}_i$).

- **Matrices**: Matrices are denoted with uppercase boldfaced letters (e.g., $\mathbf{A}$). The transpose of a matrix is denoted the same as with the vectors, with the superscript $\top$ (e.g., $\mathbf{A}^\top$).

- **Sets**: Sets are represented using uppercase calligraphic letters (e.g., $\mathcal{A}$).

2.6 Summary

In summary, this chapter has provided the necessary background information on the methodologies to be used for the research. First, various methods for synthetically modeling clouds were discussed to provide background on the benefits and weaknesses of existing techniques. Second, an overview on visual navigation methods was reviewed. Current problems with vision systems were then discussed to identify aspects for potential future improvement. Lastly, background information on stochastic modeling was explained to provide foundational knowledge for mathematical processes used in the research. The applications of these methods to the research will be discussed in Chapter III.

III. Methodology

This chapter provides an in-depth discussion of the methods used to investigate the uncertainty in a visual navigation system due to a cloudy field of view. Section 3.1 describes the process for calculating the transparency of clouds for every pixel in an image. Section 3.2 then details how the dataset of image features is created. Section 3.3 gives the model used for describing the perturbations in the data introduced by the clouds. The section then explains how the data are manipulated to make a synthetic dataset based on the original, which is then used for determining the likelihood of a mismatch between features. Lastly, Section 3.4 briefly summarizes the chapter.

3.1 Cloud Transparency Algorithm

This section describes the first step towards exploring the feature uncertainty due to clouds: developing cloud templates that can be overlaid on aerial images [26]. The algorithm aims to create a set of aerial images to which clouds can be added. It does so by calculating the transparency of each pixel in a cloud image, and is thus called the Cloud Transparency Calculation Algorithm (CTCA). The changes due to the clouds will show the effects of clouds on feature vectors. Not only must the coloring of the cloud be considered, its transparency must be preserved for partially obscuring features.

In lieu of aerial images of clouds taken from overhead, pictures of clouds are taken from directly underneath the clouds, with the assumption that the tops and bottoms of clouds are similar in appearance. As mentioned in Section 1.3, the advantage of having the sky in the background is that the background can easily be estimated to determine transparency. The semi-transparent cloud image can then be added to an

aerial image.

The setup used to collect images is given in Section 3.1.1. This section also describes the preprocessing step completed before applying the algorithm. Template creation is divided into two parts: Section 3.1.2 describes how pixels are categorized by color, and Section 3.1.3 explains how the transparency of the template is calculated.

3.1.1 Initial Setup & Preprocessing.

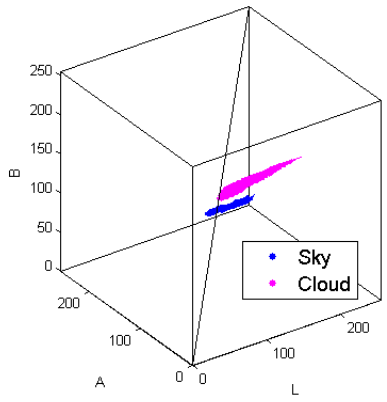
The initial setup involves extracting color information from a baseline image containing both clouds and sky. These 14-megapixel photographs were taken using a Nikon Coolpix S3100 compact digital camera. Using automatic settings, the exposure ranged from 1/1250 to 1/400 seconds.

Separate regions of cloud and sky are manually selected on the baseline image. This process is performed on a single image and then is capable of being applied to a series of test images, reducing the processing time. Figure 6 depicts the baseline image used for the research.

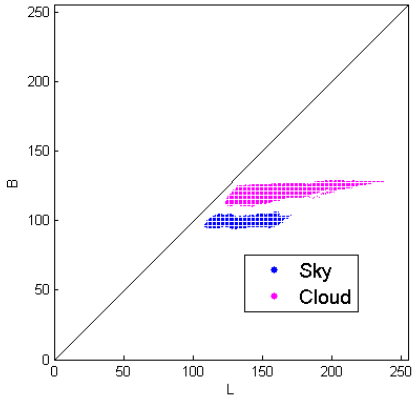
Figure 6. Baseline cloud image used for preprocessing.

For later processing, it is important that each of the regions contain only a single classification. In other words, cloud regions should not incorporate any areas of sky, and sky regions should not contain any clouds. However, both light and dark portions within the clouds should be selected for greater representation. The purpose of this segmentation is to quantitatively identify the differences between the cloud and sky regions. The segmented regions are plotted in the LAB colorspace, as seen in Figs. 7(a)-7(b). This colorspace is used as opposed to the RGB or XYZ colorspace because, out of the three, the cloud and sky segments separate the best in the LAB colorspace. A comparison of each colorspace can be seen in Fig. 7. Unlike the RGB and XYZ colorspaces, the data appear to run horizontally across the L-A plane in the LAB colorspace, allowing for easier data manipulation. For this reason, the LAB colorspace is used for the majority of the algorithm.

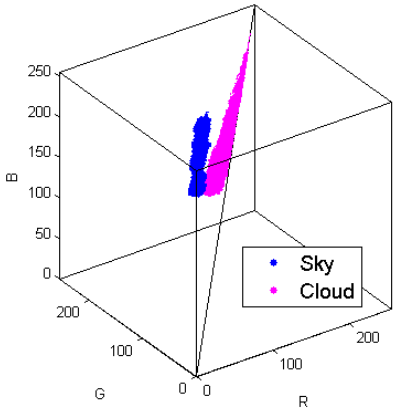
An important feature that emerges from the plotted data is that although the sky points have no defined shape to them, the cloud points have a teardrop-shaped structure. For mathematical simplicity, this teardrop shape is approximated as a cone. The cone is modeled based on its axis, endpoints, and linearly-increasing radius. The axis is determined by finding a best-fit line $\boldsymbol{\ell}$ through the cloud point data using singular value decomposition. The endpoints of the cone are identified by determining the locations of the axis that correspond to the outermost points of the cone. Lastly, the base radius of the cone is defined to be the radius of the shape at its widest point. Knowing these values, the radius at any point along the cone can be determined. Using the endpoints and the radius of the cone allows the algorithm to compute whether a pixel from the input image is inside or outside the cone, which will become useful in the following subsection. The manually-selected regions, cone endpoints, and cone radius are all saved after this step, as it will continue to be used in the algorithm, but does not need to be recomputed each time it is run.



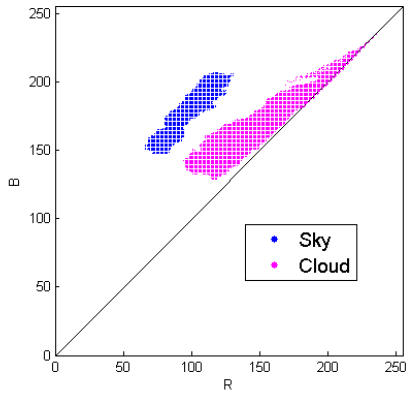
(a) LAB colorspace (rotated view)



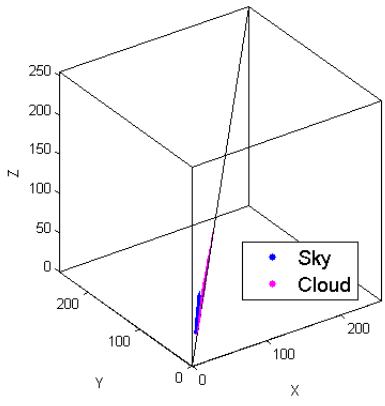
(b) LAB colorspace (L-B view)



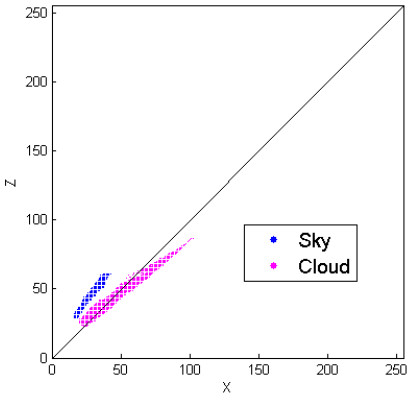
(c) RGB colorspace (rotated view)



(d) RGB colorspace (R-B view)



(e) XYZ colorspace (rotated view)



(f) XYZ colorspace (X-Z view)

Figure 7. Image data plotted in the (a)-(b) LAB colorspace; (c)-(d) RGB colorspace; (e)-(f) XYZ colorspace.

3.1.2 Color Adjustment.

The main portion of the algorithm takes an input image and classifies each pixel as either opaque or semi-transparent. An opaque pixel is one whose color matches the range of color found in dense clouds, and is found by locating pixels that lay within the modeled cone. Conversely, a semi-transparent pixel is one that resembles colors of either the sky or the partially-transparent clouds, and is located outside of the cone. These two classifications of pixels are treated differently to improve the algorithm's output. To calculate whether or not a pixel $\mathbf{u}$ is within the cone, its projection $\mathbf{u}'$ onto the axis of the cone is determined. This is done using (9):

$$\begin{aligned} \mathbf{u}' &= \left[\frac{(\mathbf{u}-\boldsymbol{\ell}_1)\cdot\boldsymbol{\ell}}{\|\boldsymbol{\ell}\|}\right]\frac{\boldsymbol{\ell}}{\|\boldsymbol{\ell}\|}+\boldsymbol{\ell}_1 \\ &= [(\mathbf{u}-\boldsymbol{\ell}_1)\cdot\boldsymbol{\ell}]\,\frac{\boldsymbol{\ell}}{\|\boldsymbol{\ell}\|^2}+\boldsymbol{\ell}_1, \end{aligned} \tag{9}$$

where $\mathbf{u}$ is the pixel being evaluated, $\boldsymbol{\ell}$ is the vector between the cone endpoints, and $\boldsymbol{\ell}_1$ is the base endpoint of the cone. Figure 8 illustrates how (9) the projection is calculated. First, the length of the projected vector is given by the dot product of $\mathbf{u}-\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}$ divided by the magnitude of $\boldsymbol{\ell}$. This scalar result is then multiplied by the unit vector of $\boldsymbol{\ell}$ (depicted by $\hat{\boldsymbol{\ell}}$) to obtain the direction of the vector. Lastly, $\boldsymbol{\ell}_1$ is added to offset the vector from the origin and into the correct location.

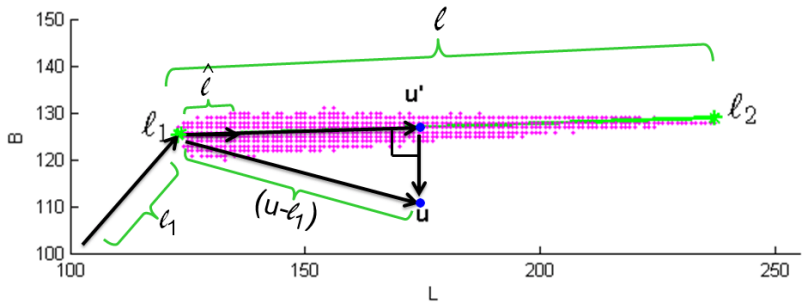


Figure 8. The vector projection of u onto $\boldsymbol{\ell}$.

After finding the projection, the difference between the point and its projection is used as the distance from the pixel to the cone. If the distance is less than or equal to the radius of the cone at the projected point, it is classified as an opaque pixel. Otherwise, it is a semi-transparent pixel. Figure 9 illustrates this process with a sample data point. The blue point located outside of the cloud points represents the single pixel that is being classified, $\mathbf{u}$. The corresponding blue point that intersects the green line is the projection of the pixel point, $\mathbf{u}'$. Because the distance between the two blue points is clearly greater than the radius of the cone at the location of the projection, the algorithm would designate this as a semi-transparent pixel.

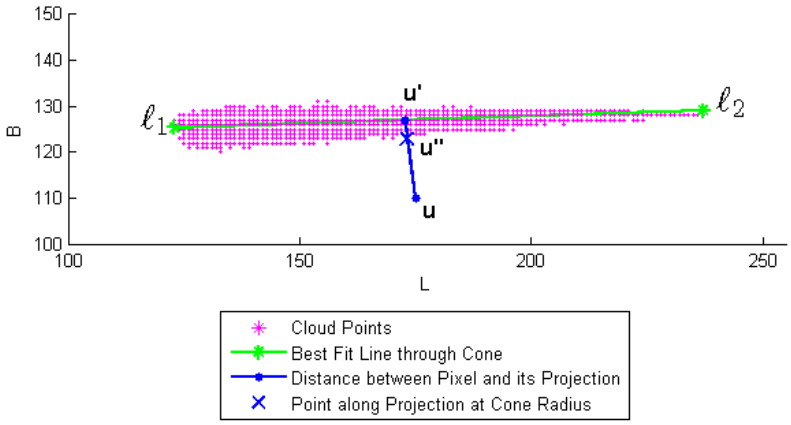


Figure 9. Sample data point projection.

The RGB color of opaque pixels remain the same as the original input image, because those pixels are already the color of the clouds. The semi-transparent pixels, however, must be altered in order to bring out their "cloudiness." To do so, each data point is moved so that it is located precisely on the outer edge of the cone. This is accomplished using (10) and (11):

$$R_{\mathbf{u}'} = \left(1 - \frac{\|\mathbf{u}' - \boldsymbol{\ell}_1\|}{\|\boldsymbol{\ell}\|}\right)\sigma \tag{10}$$

$$\mathbf{u}'' = \mathbf{u}' + R_{\mathbf{u}'} \frac{\mathbf{u} - \mathbf{u}'}{\|\mathbf{u} - \mathbf{u}'\|}. \tag{11}$$

Equation (10) computes the radius of the cone at $\mathbf{u}'$, $R_{\mathbf{u}'}$, where σ is the magnitude of the radius at the base of the cone. Equation (11) then uses that radius to determine $\mathbf{u}''$, the point between $\mathbf{u}$ and $\mathbf{u}'$ that intersects with the radius of the cone at $\mathbf{u}'$, as seen in Fig 9. The result produces an image that reduces the amount of blue tint given to the clouds where they are most transparent. This reduction is necessary so that when the cloud template is meshed with a background image, the edges of the clouds do not appear as if they are surrounded by sky. Note that when cloud images other than the baseline image are used as the input to the algorithm, the cone model of the baseline image is still used to help determine the new color calculations. This method assumes that clouds in the input image are similar in color to the clouds in the baseline image.

3.1.3 Transparency Calculation & Application.

The final step of the algorithm is determining the transparency of the cloud within the image on a pixel-by-pixel basis. To do so, two different functions are applied to the image. The first, referred to as the cloud weight function, utilizes the average LAB color of the cloud (computed from the mean of the manually-selected cloud regions from Section 3.1.1) and computes each pixel's distance from that mean point. In this case, the distance is computed by taking the norm of the difference between the current point and the mean point, and then scaling by the principal component directions:

$$x_{i,c} = \left\| \mathbf{G_c}^{-1} \mathbf{D_c} (\mathbf{u} - \bar{\mathbf{u}}_c) \right\| \frac{1}{\eta_c}, \tag{12}$$

where $x_{i,c}$ is the scaled difference, $\mathbf{u}$ is the current pixel, $\bar{\mathbf{u}}_c$ is the average LAB cloud color, $\mathbf{D}_c$ is a 3x3 matrix of the transform into principal component directions of the

manually-selected cloud region, and $\mathbf{G}_c$ is a diagonal matrix of the eigenvalues of the covariance matrix of the cloud region. η_c is an adjustment parameter to maintain the transparencies between 0 and 1.

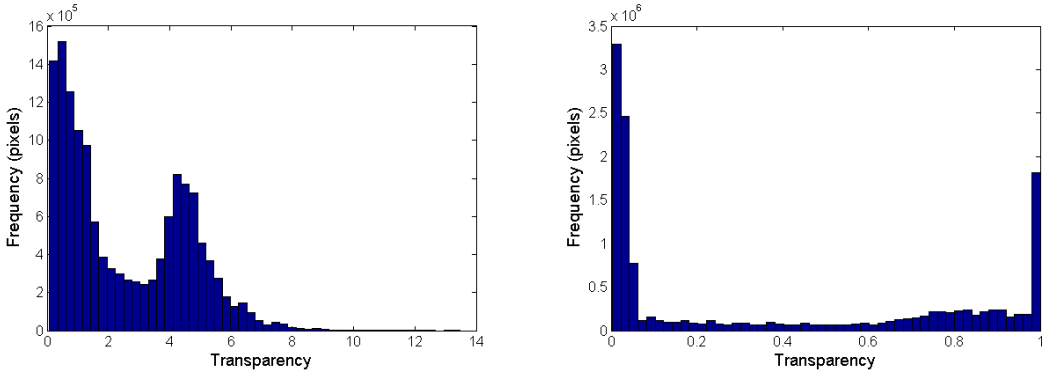
The second function, the sky weight function, performs the same operations, but with respect to the average sky color:

$$x_{i,s} = \left\| {\mathbf{G_s}}^{-1} \mathbf{D_s} (\mathbf{u} - \bar{\mathbf{u}}_s) \right\| \frac{1}{\eta_s}, \tag{13}$$

where $x_{i,s}$, $\mathbf{G_s}$, $\mathbf{D_s}$, and $\bar{\mathbf{u}}_s$ are all taken with respect to the sky.

The outputs of the functions represent how similar each pixel is to the clouds and the sky, respectively. A smaller output signifies a closer relationship, whereas a larger value indicates less similarity. The parameters η_c and η_s are used to scale the outputs such that the maximum transparency of 1 is given to pixels that most closely resemble the sky and a minimum transparency of 0 is given to those that resemble the clouds. Figure 10(a) shows a histogram of the output of the cloud weight function when applied to the baseline image before scaling by η_c. The peak near zero represents the pixels that closely resemble clouds, whereas the second peak correlates to the pixels that more closely resemble the sky. The value for η is chosen to approximate the value of the second peak in the cloud weight and sky weight functions, respectively. In this case, an appropriate value for η_c would be 5.

However, this manipulation still results in a large number of pixels with mid-range transparencies. Clouds are primarily semi-transparent along their edges, which typically occupy much less area than areas of opaque clouds or the sky. For this reason, the number of pixels with intermediate transparencies should be reduced in order to simulate real clouds. Thus, the following equation is used to provide qualitative improvement to the image by localizing the mid-range transparencies based on



(a) Original histogram (b) Final histogram (after η_c and (14) applied)

Figure 10. Histogram of the cloud weight function outputs before and after pixel transparency manipulation.

judgment:

$$x_{f,c} = \begin{cases} \frac{0.05}{\rho_c} x_{i,c} & \text{for } x_{i,c} < \rho_c \\ 0.05 + \frac{1-0.05}{1-\rho_c}(x_{i,c} - \rho_c) & \text{for } x_{i,c} \geq \rho_c \end{cases} \tag{14}$$

and from the sky weight function:

$$x_{f,s} = \begin{cases} \frac{0.05}{\rho_s} x_{i,s} & \text{for } x_{i,s} < \rho_s \\ 0.05 + \frac{1-0.05}{1-\rho_s}(x_{i,s} - \rho_s) & \text{for } x_{i,s} \geq \rho_s. \end{cases} \tag{15}$$

In (14) and (15), $x_{f,c}$ and $x_{f,s}$ represent the individual pixel transparency values after adjustment for the cloud and sky weight functions, respectively. The constants ρ_c and ρ_s are values between 0 and 1, determined experimentally. A plot of the equation using a value of $\rho_c = 0.3$ is shown in Fig. 11. When applied to all pixels of the baseline image using $\rho = 0.3$, the equation results in the histogram shown in Fig. 10(b). The output for the template is seen in Fig. 12(a).

For the sky weight function, the histogram peak near zero represents pixels that closely resemble the sky. Therefore, after using the same process as for the cloud weight function, the transparency values must be subtracted from 1 so that pixels

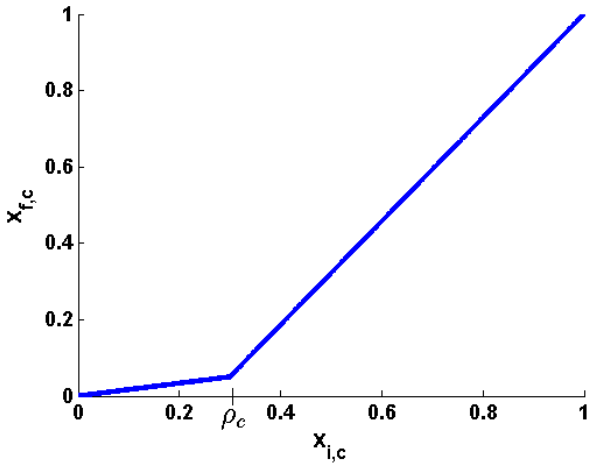


Figure 11. Equation (14) using $\rho_c = 0.3$

.

resembling sky are given high, as opposed to low, values. Figure 12(b) shows the image of final transparency values of the baseline image derived from the sky weight function, using $\rho = 0.2$ in (15). The desired output is given in Fig. 12(c), whose derivation is to follow.

Looking at the results of the two functions, it can be seen that a combination of the two are needed in order to optimize the final results. The sky weight function excels at obtaining high transparency values in the sky regions and along the edges of the clouds. However, it also gives high transparencies within shadowy portions of the clouds, which should be opaque. On the other hand, the cloud weight function produces less desirable results overall, but has lower transparencies in the shadowy cloud regions. Therefore, the optimal results should retain the high transparencies in the sky and along the cloud edges seen in Fig. 12(b), while also maintaining low transparencies within the clouds, as in Fig. 12(a).

To address this issue, a three step process is used to create a mask that helps produce the desired final transparencies. First, applying a Gaussian lowpass filter to Fig. 12(a) reduces the "speckling" effects seen in the clouds. The filter is set to have

(a) Cloud weight function transparencies

(b) Sky weight function transparencies

(c) Final template transparencies

Figure 12. Transparencies given by the: (a) cloud weight function; (b) sky weight function; (c) final output.

a standard deviation of 50 pixels over a 100-pixel block. Next, an S-curve, modeled by (16), adjusts the contrast of the image:

$$x_{s,c} = 0.5 + \frac{1}{\pi}\tan^{-1}\left(\frac{x_{f,c} - 0.35}{0.05}\right), \tag{16}$$

where $x_{s,c}$ represents the contrast-adjusted values of $x_{f,c}$. This adjustment suppresses transparency values in the clouds and retains those in the sky. As depicted in Fig. 13, dividing by π constrains the S-curve outputs to be between -0.5 and 0.5, so the 0.5 is added to keep the transparency values between 0 and 1. Subtracting 0.35 from $x_{f,c}$ controls the inflection point on the s-curve, while dividing by 0.05 gives the steep

slope seen around the inflection point.

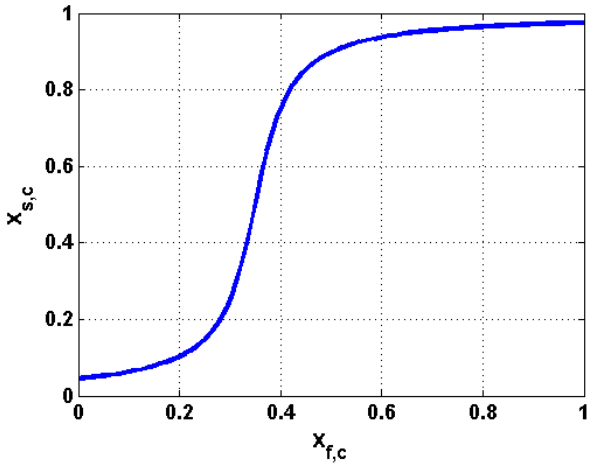


Figure 13. Contrast adjustment S-curve.

Lastly, the image is dilated so that the edges of the dark cloud regions recede slightly. When this cloud weight mask is applied to the sky weight function output, receding the edges of the mask allows the edge values from Fig. 12(b) to remain, as desired. The results of the cloud weight mask applied to the sky weight output is seen in Fig. 12(c).

Notice that the majority of the cloud regions are very dark, corresponding to nearly full opacity in the final output image. Around the edges of the clouds, the color turns grey, indicating that some of the color from the input image will remain in the output, but will be combined with the color from the background image. Lastly, the sky areas are primarily white. The white signifies that this portion of the image is almost or completely transparent, allowing the background image to come through.

The greyscale transparency image is applied to the cloud input image and the desired aerial image using element-wise multiplication denoted by $\odot$:

$$\mathbf{I}_o = \mathbf{T} \odot \mathbf{I}_g + (1 - \mathbf{T}) \odot \mathbf{I}_c \ , \tag{17}$$

where $\mathbf{T}$ is the transparency image, $\mathbf{I}_g$ is the background aerial image, $\mathbf{I}_c$ is the input cloud image, and $\mathbf{I}_o$ is the final output image. The resulting image retains the clouds of the input and overlays them on top of the background, while maintaining realistic transparency within the cloud. Figure 14 displays the final result of the algorithm, using the baseline image from Fig. 6 as the input overlaid on an aerial image [27].

Figure 14. Resulting output of the cloud transparency calculation algorithm using baseline image as input.

Section 4.1 demonstrates results using several different cloud images as baseline images and evaulating the outputs resulting from multiple input images. By completing this analysis, the effectiveness of the algorithm can be evaluated in two ways. First, it can be determined if the algorithm is flexible to using other baseline images to create the cone model, or if the model was valid only for the particular image shown in Fig. 6. Secondly, the analysis will help determine if an image other than the baseline image can be used as the input image to the algorithm. In other words, it will determine if a cone model must be determined on an individual basis, or if a single model is sufficient to calculate transparency values for a wide range of cloud images.

3.2 Feature Keypoint & Descriptor Generation

Using the cloud templates generated with the algorithm described in Section 3.1, numerous cloudy aerial images are created. These images will hereafter be referred to as "test" images, whereas their corresponding original aerial images will be referred to as "truth" images. The following discussion describes the process with which the feature vectors to be studied are generated.

3.2.1 Image Dataset Creation.

To create a database of images, a specified cloud image is first input to the CTCA to create a cloud template. The resulting cloud template is then applied to the desired background image. The background image used for the research was obtained from the USGS EarthExplorer website and is shown in Fig. 15 [27]. This photograph

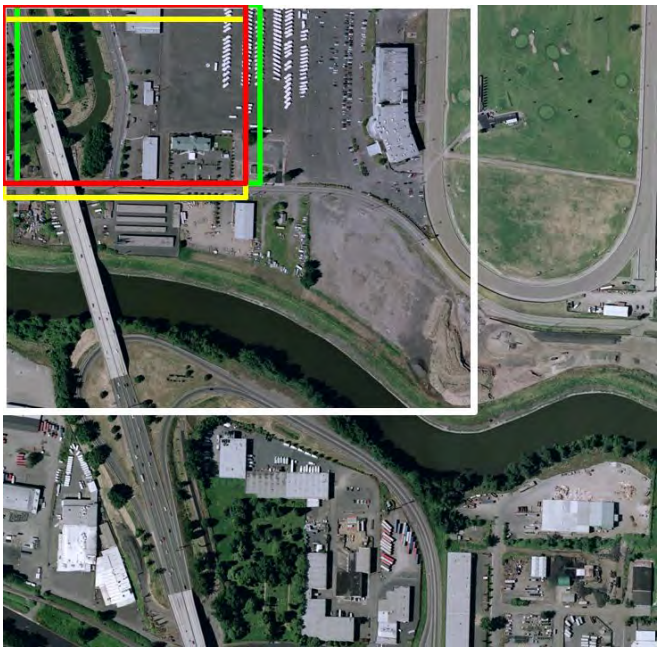


Figure 15. Background aerial image used for analysis. The region within the white box is subdivided into 400 different overlapping regions to simulate a motion sequence. The colored boxes show the amount of overlap between images.

depicts an area of the Portland, Oregon metropolitan area in 2005. A high-resolution

photograph was selected so that the image can be segmented into smaller subregions. The region within the white box shows the full extent of the image used for finding features. This region is subdivided into a 20x20 image array, resulting in 400 different images used for analysis. Each image is 300x400 pixels in size and is offset from its neighboring images by 100 pixels in both the vertical and horizontal directions to simulate a motion sequence. The amount of vertical and horizontal offset is depicted by the colored boxes in the figure. Each of the raw segmented images are referred to as the "truth images." After the images have been segmented, a 300x400 portion of the cloud template is selected and overlaid onto each of the sub-images. The resulting images are the "test images." Figure 16 demonstrates one such test image, along with its corresponding truth image. This portion of the cloud template was chosen due to the roughly equal distribution of cloudy and non-cloudy regions within the image.

(a) Truth image (b) Test image

Figure 16. (a) A sample truth image and (b) its corresponding test image.

3.2.2 Feature Detection.

The images created in Section 3.2.1 are used to detect features. The SURF algorithm, as described in Section 2.2.1.4, is first used to find features on the truth images. The algorithm finds the strongest features in the image, where the specific

number of resulting features is partially dependent on the threshold value applied. For this research, many feature points are desired so that a large amount of data can be collected, corresponding to relatively low input thresholds. The lower the threshold used, the more features are found in an image. This occurs at the expense of the quality of the features, however, so this introduces a limitation on the research. If the threshold selected is too small, it may not reflect realistic feature selections for a visual navigation system. For this research, the threshold was chosen so that a minimum of 100 features were detected on each image.

After using SURF to generate the strongest features for the truth images, SURF is used once again to find corresponding features on the test images. However, as mentioned in Chapter I, the clouds on the test images affect the placement of the feature locations. In other words, a feature found in a truth image is not guaranteed to be found in the same location or even found at all if it has been partially or fully obscured by clouds in the test image. Therefore, features on the test images are forced to be in the same locations as those in the truth images. This is done to show how the feature changes with the introduction of clouds into an aerial image, but also generates a limitation on the research for the reasons stated above.

To force matches, the feature keypoints from truth images are input to SURF before finding features for the test images. By inserting keypoints, the algorithm skips the feature detection stage and calculates the feature descriptors only at the provided (x, y) coordinates. This forced operation allows for a comparison between features that are guaranteed to match without spatial errors. Forcing matches based on the keypoint location also forces the keypoint's size, response, octave, and class ID to be the same as the original keypoint on the truth image. The only keypoint property that changes is the angle of orientation, which is calculated using the feature descriptor. The feature descriptors on the test images are not identical to those from

the truth images, since the calculation of the descriptor is dependent on the image content, which changes when the clouds are overlaid. 40,000 matching features are collected from the truth and test images: 100 from each pair of the 400 corresponding images. The changes in the descriptors between the truth and test images is the basis for modeling the error introduced by the clouds and is discussed in Section 3.3.1.

3.2.3 Transparency-Based Feature Categorization.

Simply finding features between the truth and test images is not enough preparation for analyzing the data. Within each of the test images, there are three distinct regions. First, there are regions that purely show the ground. These are regions in which the calculated cloud transparencies have a value near 1, meaning that no clouds are present in that portion of the image. The second type of region is the edges of the clouds. These regions have transparencies that range from just above 0 to just below 1. Lastly, there are portions of the clouds that are purely cloud without any edges. These regions correspond to transparencies near 0. To more accurately describe the features, each feature is categorized based on the region in which the detected feature was located. The three regions are referred to as clear, edge, and cloud regions, respectively.

The features are mathematically categorized based on the average transparency of each feature. The magnitude of the radius, as specified by the keypoints, determines the circular area over which the average transparency value is calculated for a particular feature. Figure 17 depicts the average transparency values for every test feature. The cutoff values for the transparencies are experimentally determined to be 0.05 and 0.93, depicted as the dashed lines in Fig. 17. Thus, any feature with an average transparency less than 0.05 is deemed a "cloud feature", any with an average transparency greater than 0.93 is determined to be a "clear feature", and features

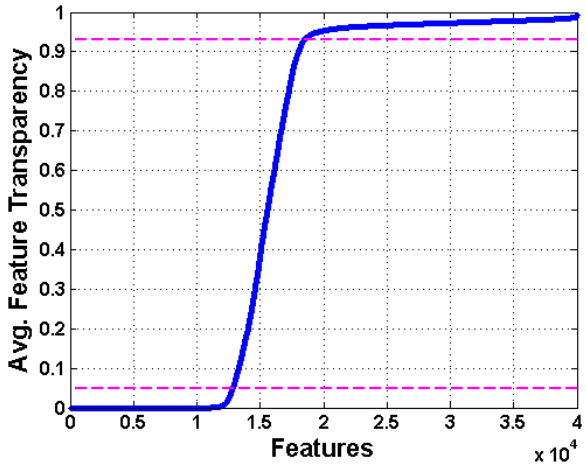


Figure 17. The average transparency calculated for every feature, sorted from lowest to highest.

with intermediate average transparencies are referred to as "edge features." Because the average transparency is the basis for categorizing features, it must be noted that features along the boundary of abrupt transparency change (e.g. from 0 to 1) and features with a constant mid-range transparency (e.g. 0.5) will both be categorized as edge features. This is a known limitation of the feature categorization process. Table 1 displays the number of features found for each of the categories.

Table 1. Feature Categorizations

	Cloud	Edge	Clear
Transparency Region	"Low"	"Medium"	"High"
Transparency Range	[0, 0.05)	[0.05, 0.93]	(0.93, 1]
Number of Features	12,842	5,620	21,538

All further analysis on the features are split into the three categories so that the features can be more accurately and usefully analyzed. Without the categorization, results would be meaningless because each region has different characteristics. For example, the edges of objects tend to be strong contenders for features, so by separating the edges of the clouds into their own region of analysis, it can be determined

if the edges add a bias to the feature vectors. The clear regions, on the other hand, should change very little between the truth and test images. Lastly, the effects on the cloud regions are unrelated to the original feature, but should be significant based on the large differences between the truth and test images in these regions.

3.3 Development of Feature Uncertainty

The aim of the research is to find the probability of mismatch between two features located in the same image. To find this probability, several steps are taken. First, an appropriate model for describing the error introduced by the clouds is generated. Principal component analysis is then used for reducing the dimensionality of the feature descriptors. Next, a random sampling approach is used to build up a synthetic dataset based on the statistics of the transformed error vectors. After expanding the dimensions of the data back to the original size, Monte-Carlo methods are employed to find the probability of a mismatch. The following sections provide a detailed explanation of required steps.

3.3.1 Modeling Cloud-Induced Feature Error.

In order to quantify the feature error due to the clouds, an appropriate model must be composed. Three different error models were initially tested, but only one was suitable for describing the dataset. The following discussion describes the chosen error model; a description of the other two models and their shortcomings are discussed in Chapter IV.

The following model is used to characterize the error between truth and test images:

$$\mathbf{v}_{noised} = \mathbf{v}_{truth} + \boldsymbol{\epsilon}. \tag{18}$$

In (18), $\mathbf{v}_{truth}$ represents a single descriptor from a truth image, $\mathbf{v}_{noised}$ represents

the perturbed version of $\mathbf{v}_{truth}$ after clouds have been added, and $\boldsymbol{\epsilon}$ is the unknown probabilistic error between the two descriptors.

To determine $\boldsymbol{\epsilon}$, first each individual truth descriptor is subtracted from its corresponding test descriptor. The differences in these descriptors are referred to as the "error vectors," which represent the error caused by the clouds. Each of the error vectors are placed into the same category as the test descriptor used to generate it.

3.3.2 Dimensionality Reduction.

Principal component analysis (PCA) is used to reorganize data by the amount of variance present. Oftentimes, variables (dimensions) within a set of data are correlated to one another. A perfect correlation between multiple variables implies that only a single variable is needed in order to characterize the others. While perfect correlation is rare, high correlation between variables still allows a single variable to largely characterize the others. Therefore, performing PCA on a high-dimensional dataset allows one to reduce the dimensionality of the data without much loss to the informational content of the original data. To obtain the principal component data from the original data, the following equation is used:

$$\mathbf{s}_{pca} = (\mathbf{s}_{orig} - \boldsymbol{\mu}_{orig}) * \mathbf{C} \ , \tag{19}$$

where $\mathbf{s}_{pca}$ is the 1xN data in principal component space, $\mathbf{s}_{orig}$ is the 1xN data in the original space, $\boldsymbol{\mu}_{orig}$ is a vector of means for each of the N dimensions in $\mathbf{s}_{orig}$, and $\mathbf{C}$ is the NxN transformation matrix of the MxN data in the original space. M represents the number of observations in a dataset, while N represents the number of dimensions from the original set of error vectors, which is 64 for SURF features.

PCA is implemented on each of the three types of error vectors described in Section 3.2.3: clear, edge, and cloud. After performing PCA, only the first P dimensions

are retained; the remainder of the data are permanently rejected. P is determined by looking at the percentage of variance of the original 64-dimensional data is retained in the P-dimensional data. A small value of P would not capture the nature of the data, while a large value would defeat the purpose of reducing the dimensionality of the data. Figure 18 shows the percent variance retained for the first 20 dimensions of the error vectors in the PCA space. The plot shows that for each region, approximately

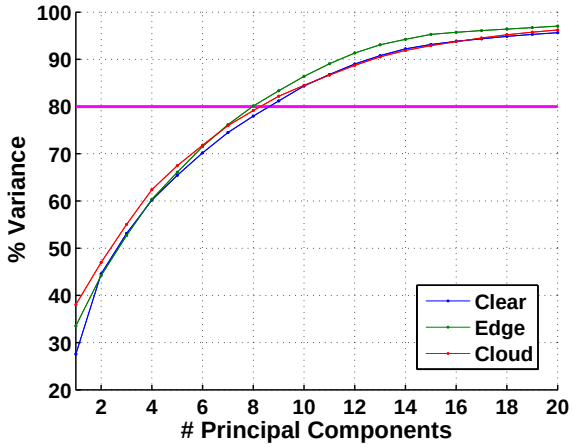


Figure 18. Percent variance retained for the first 20 dimensions of the error vectors in the PCA space.

80% of the variance in the original data is retained within the first 8 dimensions of the transformed data. Therefore, P is chosen to be 8. The last 64-P dimensions of the transformed data are then permanently rejected. However, the vector of means and the coefficient matrix from the original data, as described in (19), are stored for future use. The error vectors in the PCA space reduced to P dimensions are referred to as the "transformed error vectors."

3.3.3 Synthetic Dataset Creation.

To avoid dependence of the results on a fixed dataset, a synthetic dataset is created based on the fixed dataset of the transformed error vectors. The synthetic dataset is designed to be statistically similar to the transformed dataset by utilizing the inverse

of the CDFs of the transformed error vectors. A random sampling approach is taken to pass a random number X through the inverse CDFs of the transformed dataset to determine the corresponding value x, such that $F(x) = X$. The following discussion provides a detailed description of the process.

The method for creating the synthetic dataset selects a uniformly-distributed random number on the interval [0,1] that passes through the inverse of the CDF of the transformed dataset. This random number represents a probability threshold, such as 0.5 producing the median, and 0.25 producing the lower quartile. Each random input representing a cumulative probability produces a corresponding output associated with that probability. The output is the result of linearly interpolating between the accepted set of data at the value specified by the random input. The interpolated output is a single resulting sample for the synthetic dataset. This process is repeated for each dimension within the transformed dataset. Through this process, it is determined which points in the CDF are within a given threshold of the random number, and data that do not fall within the threshold are rejected. For each subsequent dimension, only data that were accepted for each previous dimension are used for the threshold comparison. This implies that the CDF is only produced from the data in the dataset that is consistent with previously-generated values. The number of repetitions of the process corresponds to the desired number of observations for the synthetic dataset.

Pseudocode for the synthetic data generation process is given in Algorithm 1. The algorithm keeps track of the "accepted" indices of the data that are retained from each dimension, given by $\mathcal{A}$. Since the process is dependent on previous outputs, the accepted indices for the first dimension are initialized to include the entire set of the transformed data in the first dimension. The transformed data is given by $\mathbf{F}$. The "threshold" indices are the indices for which the magnitude of the difference between

the p^{th} dimension of $\mathbf{F}$ and the interpolated output $\boldsymbol{\phi}$ is less than the threshold t. Threshold indices are denoted as $\mathcal{T}$. The accepted indices are updated for each dimension, based on the intersection of the indices that have previously been accepted and the threshold indices for the p^{th} dimension. The pseudocode demonstrates the data generation process for a single point. To generate a synthetic dataset with M observations, the algorithm must be repeated M times. Subscripts of the variables denote the dimension of the variable.

$\mathcal{A}_1$ = all indices of $\mathbf{F}_1$;
$\boldsymbol{\phi}_1$ = interpolation of a uniformly-sampled random number using
value of $\mathbf{F}_1(\mathcal{A}_1)$ on the interval $[0, 1]$;
for *dimension* p = *2:P* **do**
  $\mathcal{T}_p = |\mathbf{F}_{p-1} - \boldsymbol{\phi}_{p-1}| < t$;
  $\mathcal{A}_p = \mathcal{A}_{p-1} \cap \mathcal{T}_p$;
  $\boldsymbol{\phi}_p$ = interpolation of a uniformly-sampled random number using
  values of $\mathbf{F}_p(\mathcal{A}_p)$ on the interval $[0, 1]$;
end

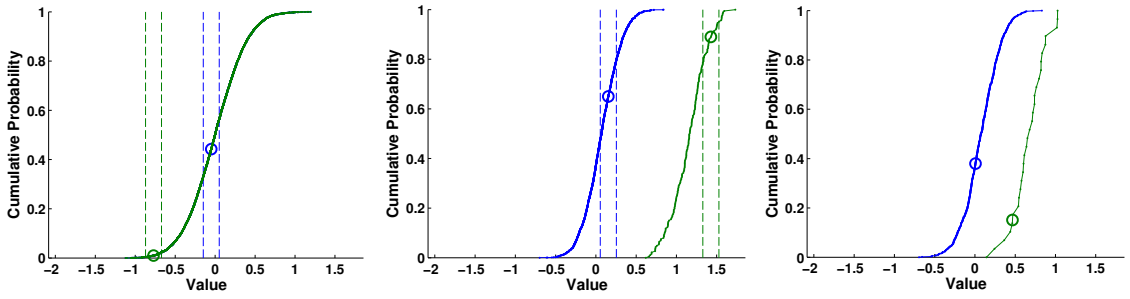
Algorithm 1: Synthetic Data Generation

To illustrate this process, an example is provided using correlated Gaussian noise in 3 dimensions. The blue and green data points depict the process for two separate runs of the algorithm. First, the CDF of $\mathbf{F}_1(\mathcal{A}_1)$ is plotted in Fig. 19(a). In the first dimension, the accepted indices $\mathcal{A}_1$ consist of all the indices of $\mathbf{F}_1$. A uniformly-sampled random number is then generated, which is given in Fig. 19(a) as the y-value of the colored circle. The first random number is 0.44 for the blue run and 0.01 for the green run. The returned value for the synthetic dataset in the first dimension, $\boldsymbol{\phi}_1$, is then given by the interpolated value along the CDF corresponding to the x-value of the colored circles. The outputs are given as -0.05 and -0.77 for the blue and green runs, respectively.

For the second dimension, the threshold indices $\mathcal{T}_2$ must first be found. $\mathcal{T}_2$ is determined by identifying the indices where $|\mathbf{F}_1 - \boldsymbol{\phi}_1| < t$, where $t = 0.1$. Data points

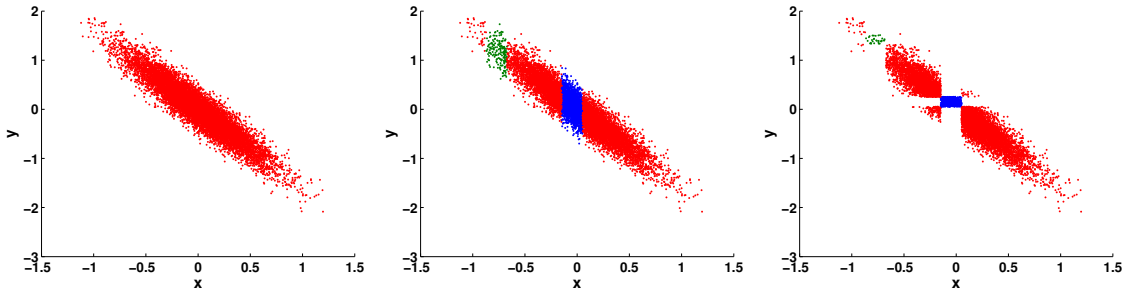
falling within this threshold are depicted within the vertical dashed lines in Fig. 19(a). The accepted indices $\mathcal{A}_2$ are then determined by the intersection of $\mathcal{A}_1$ indices and $\mathcal{T}_2$ indices. After determining $\mathcal{A}_2$, the CDF of only the data in the second dimension that fell within the threshold and was also a previously accepted index, $\mathbf{F}_2(\mathcal{A}_2)$, is plotted. These particular CDFs for the blue and green runs are shown in Fig. 19(b). Next, another random number is selected along this CDF, shown in Fig. 19(b) as the y-value of the colored circles. The output $\boldsymbol{\phi}_2$ is once again given as the interpolated value along the CDF corresponding to the x-value of the colored circles. From the random number inputs of 0.65 and 0.89, the outputs for the second dimension of the synthetic dataset are determined to be 0.15 and 1.43 for the blue and green runs, respectively.

The process is again repeated for the third dimension. The threshold indices $\mathcal{T}_3$ are found as the indices where $|\mathbf{F}_2 - \boldsymbol{\phi}_2| < t$ and used to find the accepted indices $\mathcal{A}_3 = \mathcal{A}_2 \cap \mathcal{T}_3$. Figure 19(c) shows the CDFs of the data in the third dimension that fell within the threshold and was also a previously accepted index, $\mathbf{F}_3(\mathcal{A}_3)$. The interpolated outputs for the third dimension of the synthetic dataset, using new random numbers of 0.38 and 0.15, are given as 0.01 and 0.47 for the blue and green runs, respectively. After reaching this point, the algorithm is completed and must be rerun to generate additional points for the synthetic dataset.



(a) Dimension 1 (b) Dimension 2 (c) Dimension 3

Figure 19. Example of a Gaussian dataset creation in 3 dimensions. The blue and green data points correspond to two separate runs of the algorithm.



(a) $\mathbf{F}(\mathcal{A}_1)$ (b) $\mathbf{F}(\mathcal{A}_2)$ (c) $\mathbf{F}(\mathcal{A}_3)$

Figure 20. A representation of the accepted Gaussian data through the dimensions. The blue and green data points correspond to the two separate runs of the algorithm.

Figure 20 shows the effects of the algorithm on the 3-dimensional Gaussian data. The red data points depict the Gaussian data of the accepted indices for each dimension, $\mathbf{F}(\mathcal{A}_p)$. The blue and green data points correspond to the same two runs of the algorithm as depicted in Fig. 19. As given in Algorithm 1, the accepted indices for the first dimension consist of all the indices of $\mathbf{F}$. Thus, the full dataset is shown in Fig. 20(a). For the second dimension in Fig. 20(b), the points are narrowed down to only the blue and green data, which represents $\mathbf{F}(\mathcal{A}_2)$ for the blue and green runs, respectively. Lastly, Fig. 20(c) shows the resulting Gaussian data $\mathbf{F}(\mathcal{A}_3)$ after proceeding to the third dimension. Note that for the second dimension, the Gaussian data was thresholded in the x-direction, while for the third dimension, it was thresholded in both the x- and y-directions. This shows how the data for the current dimension is dependent on thresholding from each of the previous outputs.

As previously explained, Algorithm 1 relies on linear interpolation to produce the outputs $\boldsymbol{\phi}_p$. It is important to note that linear interpolation requires two or more points to interpolate between. Thus, special handling is required for the case in which there are not enough accepted indices of the data to allow interpolation. This special handling is broken up into two possible scenarios. If interpolation is attempted when there are zero accepted indices, the algorithm bypasses the interpolation and instead assigns $\boldsymbol{\phi}_p$ to be zero. If, however, one to two data points are present, one of the data

points is randomly selected and assigned as the output of the interpolation. Even though linear interpolation is possible between two points, it has the possibility of introducing new data points that are not present in the original data, as in the case of a dataset with a bimodal distribution. Therefore, interpolation is only used when three or more accepted indices are present.

By repeating the algorithm M times, many different possible values occur, which gradually builds up a synthetic dataset. Three different synthetic datasets are created for each of the clear, edge, and cloud transformed error vectors. The statistical similarity of the synthetic datasets to the transformed datasets is investigated in Section 4.3.

3.3.4 Dimensionality Expansion.

Upon generating datasets based on the transformed error vectors, the synthetic dataset must be projected back into the original 64-dimensional space so that they may be compared with the original feature descriptors from the truth and test images. Equation (20) demonstrates the process by which this is achieved.

$$\mathbf{s}'_{orig} = \mathbf{s}_{pca} * (\mathbf{C}')^{\top} + \boldsymbol{\mu}_{orig} \tag{20}$$

In (20), $\mathbf{s}'_{orig}$ represents the 1xN synthetic data in the principal component space projected back into the original space, and $\mathbf{C}'$ represents the PxN sub-array of $\mathbf{C}$ corresponding to the first P dimensions. $\mathbf{s}'_{orig}$ does not completely match $\mathbf{s}_{orig}$ because of the $N - P$ dimensions of data that are discarded when reducing dimensions. Therefore, the following equations are used in order to re-introduce noise into the $(P+1)^{th}$ through N^{th} dimensions of the error vectors:

$$\boldsymbol{\omega} = \boldsymbol{\omega}_g * \mathbf{G''} * (\mathbf{C''})^{\top} \tag{21}$$

$$\boldsymbol{\epsilon}' = \mathbf{S}'_{orig} + \boldsymbol{\omega}. \tag{22}$$

In (21), $\mathbf{C}''$ is the Nx(N-P) sub-array of $\mathbf{C}$, $\mathbf{G}''$ is the (N-P)x(N-P) sub-array of the diagonal matrix of the square root of the eigenvalues of the covariance matrix, and $\boldsymbol{\omega}_g$ is a (N-P)x1 matrix of zero-mean Gaussian noise. In (22), the noise is added on to the synthetic data in the principal component space to form $\boldsymbol{\epsilon}'$. $\boldsymbol{\epsilon}'$ is termed the "synthetic error vectors," which are used for the remainder of the data analysis.

3.3.5 Probability of Mismatch.

Finally, Monte-Carlo techniques are employed to find the probability of mismatches between features. As explained in Section 2.4.2, the Monte Carlo method allows for the empirical determination of a statistic of interest. For this research, the interest is in finding the probability of a mismatch. More specifically, a mismatch is defined as a feature on an image that more closely matches a different feature on the same image than it does to itself, after the synthetic error vectors are introduced into the two compared features. In mathematical terms, this probability can be described as such:

$$\begin{aligned} \text{mismatch}_{\text{a,b}} &= \|\mathbf{v}_a - (\mathbf{v}_a + \boldsymbol{\epsilon}'_i)\| > \|\mathbf{v}_a - (\mathbf{v}_b + \boldsymbol{\epsilon}'_j)\| \\ &= \| - \boldsymbol{\epsilon}'_i\| > \|\mathbf{v}_a - \mathbf{v}_b - \boldsymbol{\epsilon}'_j\| \end{aligned} \tag{23}$$

$$\Pr\{\text{mismatch}_{\text{a,b}}\} = \frac{1}{k_\epsilon}\sum_{a=1}^{k_\epsilon} \text{mismatch}_{\text{a,b}}. \tag{24}$$

Equation (23) describes the relationship that must be true in order for feature b to be mismatched to feature a. $\mathbf{v}_a$ and $\mathbf{v}_b$ represent two different feature descriptors from a single truth image and $\boldsymbol{\epsilon}'_i$ and $\boldsymbol{\epsilon}'_j$ represent two different randomly-selected synthetic error vectors generated in Section 3.3.4. Equation (24) then sums the number of

identified mismatches, where k_ϵ gives the number of times random selections of $\boldsymbol{\epsilon}'_i$ and $\boldsymbol{\epsilon}'_j$ are made per feature comparison. To convert this value into a probability, the result is then divided by k_ϵ. If two different features on the truth image are identified, this calculation will determine the likelihood that feature b is incorrectly identified as feature a when the clouds are introduced in the corresponding test image, when both features are located within the same transparency region.

In order to obtain an approximation that represents the entire population of data, (23) is repeated many times. Every feature a calculated on the test images is compared to every other feature b. Additionally, for every comparison, numerous random selections of $\boldsymbol{\epsilon}_i$ and $\boldsymbol{\epsilon}_j$ are made in order to obtain a better representation of the average outcome. To maintain reasonable computational speeds, 1,000 features from each category are selected to compare amongst each other and 10,000 synthetic error vectors are randomly selected for each feature pair from the pool of $1\text{x}10^6$ total synthetic error vectors. As with before, the calculation of the probability of a mismatch is segmented into the clear, edge, and cloud features. The outcome of the trials are detailed in Chapter IV.

3.4 Summary

In summary, this chapter has thoroughly described the methodology used in order to accomplish the research goals. The first stage of the research included development of an algorithm to determine cloud transparency. The second stage used the algorithm to generate features between cloud-free and cloudy aerial images. Subtracting the features between the two types of images allowed for the investigation of the errors in feature descriptors due to clouds. Next, the third stage of the research produced a synthetic dataset that is statistically-similar to the dataset of transformed error vectors. Lastly, this synthetic dataset was used to determine the probability

of mismatches between features. The results of each stage of research are shown in Chapter IV, as well as an evaluation of how statistically-similar the synthetic dataset is to the transformed error vectors.

IV. Results & Analysis

After implementing the methodologies in Chapter III, the results are generated and analyzed. Section 4.1 discusses the results of the CTCA for cloud template creation. To ensure that the algorithm is not dependent on any one particular choice of baseline image, three different baseline images are chosen and the results from each are compared. Section 4.2 then explains the two rejected methods for modeling the error introduced into the features by clouds. The section provides proof and explains the rationale behind eliminating these two methods as valid error models. The results of the generation of the synthetic dataset are shown in Section 4.3 for each of the ground, edge, and cloudy feature regions. Section 4.4 then describes a numerical approach to evaluating the statistical similarity of the synthetic error vectors to the original error vectors to ensure validity of results using the synthetic error vectors. Next, the results of the Monte-Carlo simulations that describe the probability of a mismatch between features are discussed in Section 4.5. Lastly, a summary of the chapter is given in Section 4.6.

4.1 Cloud Transparency Calculation Algorithm

The basis for much of the analysis completed for the research depends on the integrity of the cloud transparency algorithm. Not only must the algorithm be able to produce realistic cloud overlays for clouds with varying structures, but it ideally must also be able to do so for arbitrary baseline images. This section tests the effectiveness of the CTCA using three different baseline images, each of which calculate the transparency of four different input images. The baseline images calculate the approximated cone model that determines which pixels belong to cloud regions and which belong to sky regions. Therefore, using different baseline images will prove

that the cone model approximation in Section 3.1.1 is valid for images of clouds with various structures. The input images are the images used for transparency calculations and cloud template creation. Testing input images that differ from the baseline image ensures that using a single baseline image is sufficient to generate many cloud templates.

4.1.1 Analysis Setup.

The three selected baseline images are also used as three of the input images. A single additional image is used as an input image only. Figures 22(a), 23(a), and 24(a) depict the three images used as both baseline and input images, while Fig. 25(a) shows the additional input image. Each baseline image is used to produce templates for each input image, for a total of 12 generated cloud templates. The images are chosen to represent a range of possible cloud structures.

Equations (12) through (15) in Section 3.1.3 gave expressions with experimentally-determined parameters. These parameters include η_c, η_s, ρ_c, and ρ_s. The values for η_c and η_s are determined by selecting the approximate value of the second peak in the histograms that describe the cloud similarity and sky dissimilarity of each pixel, such as the one shown in Fig. 10 in Section 3.1.3. The values of ρ_c and ρ_s are then used in (14) and (15) to help reduce the amount of pixels with intermediate transparencies for each histogram. Table 2 lists the chosen values for each pair of baseline and input images. Note that η changed while ρ remained constant, representing slight differences in color distributions between images.

4.1.2 Algorithm Results.

After setting the parameters of the CTCA, the remainder of the algorithm proceeds as described in Section 3.1.3 without any additional adjustments. Figure 21

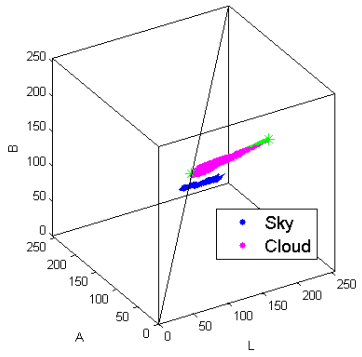
Table 2. Transparency Calculation Parameters

Baseline Image	Input Image	η_c	η_s	ρ_c	ρ_s
DSCN3536	DSCN3536	4	25	0.3	0.2
	DSCN3635	4	17	0.3	0.2
	DSCN3522	4	18	0.3	0.2
	DSCN3548	4	15	0.3	0.2
DSCN3635	DSCN3536	10	50	0.3	0.2
	DSCN3635	12	30	0.3	0.2
	DSCN3522	8	55	0.3	0.2
	DSCN3548	5	40	0.3	0.2
DSCN3522	DSCN3536	16	12	0.3	0.2
	DSCN3635	10	15	0.3	0.2
	DSCN3522	10	15	0.3	0.2
	DSCN3548	16	12	0.3	0.2

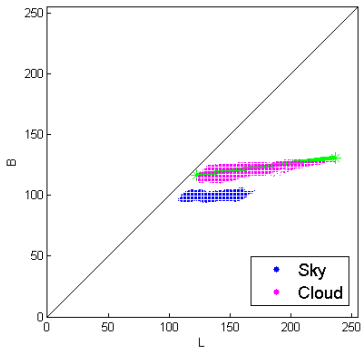
shows the plotted cloud points for each baseline image. The cone approximations are based off the cloud points, where the calculated cone endpoints are shown as green stars, and the axis of the cone is shown as a green line. The cone parameters are entirely calculated from the algorithm; no manual input is involved. As seen from the figures, the structure of the plotted cloud points changes with respect to each baseline image.

After the algorithm uses the baseline images to construct a cone model, categorize the pixels, and calculate the transparency, the cloud overlays are generated. Figures 22 through 25 depict the overlays of the three different cloud templates when applied to the aerial image from Fig. 15. The results of the cloud overlays show that, overall, the algorithm performs well for each combination of baseline and input image. Even though the exact shape of the cloud pixels in the LAB colorspace change for each baseline image selection, the cone model approximation works well enough to be able to successfully differentiate the cloud pixels from the sky pixels, which is reflected in the cloud overlays. However, some of the edge regions are given lower transparency values than is ideal. For example, in Fig. 23(c), some of the calculated transparency values along the cloud edges are lower than necessary, which results in

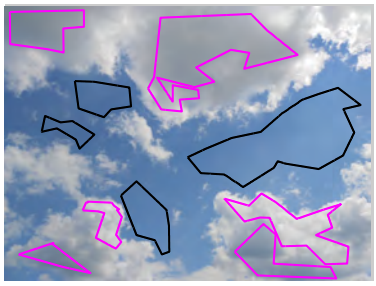
(a) DSCN3536 selected regions



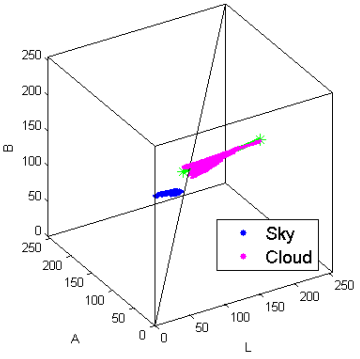
(b) DSCN3536 plotted regions (rotated view)



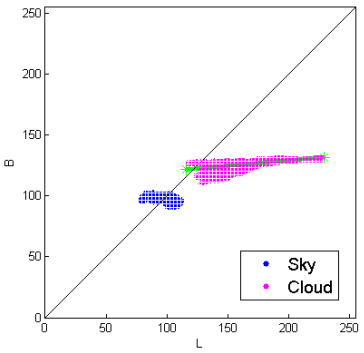
(c) DSCN3536 plotted regions (L-B view)



(d) DSCN3635 selected regions

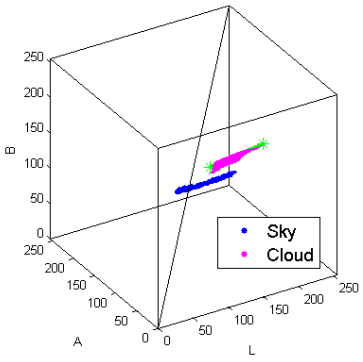


(e) DSCN3635 plotted regions (rotated view)

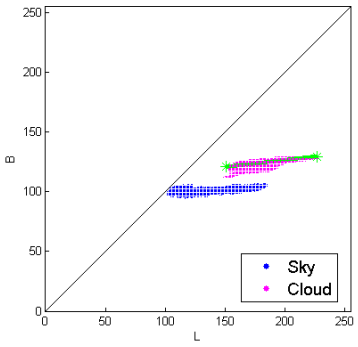


(f) DSCN3635 plotted regions (L-B view)

(g) DSCN3522 selected regions



(h) DSCN3522 plotted regions (rotated view)



(i) DSCN3522 plotted regions (L-B view)

Figure 21. Selected cloud and sky regions, along with their representation in the LAB colorspace. The baseline images are: (a)-(c): DSCN3536, (d)-(f): DSCN3635, (g)-(i): DSCN3522.

(a) Input image: DSCN3536

(b) Results after applying input image to baseline image DSCN3536

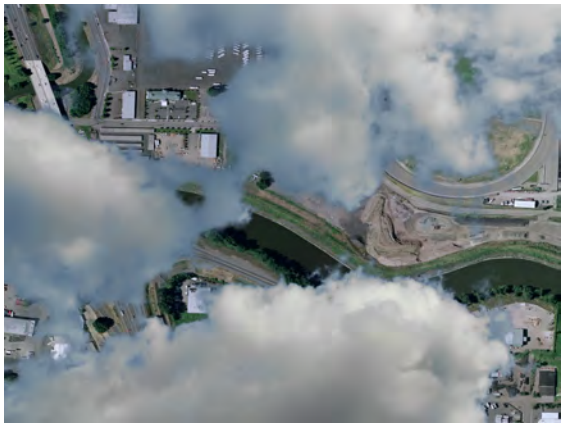
(c) Results after applying input image to baseline image DSCN3635

(d) Results after applying input image to baseline image DSCN3522

Figure 22. Resulting outputs using input image (a) DSCN3536 for the baseline images: (b) DSCN3536; (c) DSCN3635; (d) DSCN3522.

(a) Input image: DSCN3635

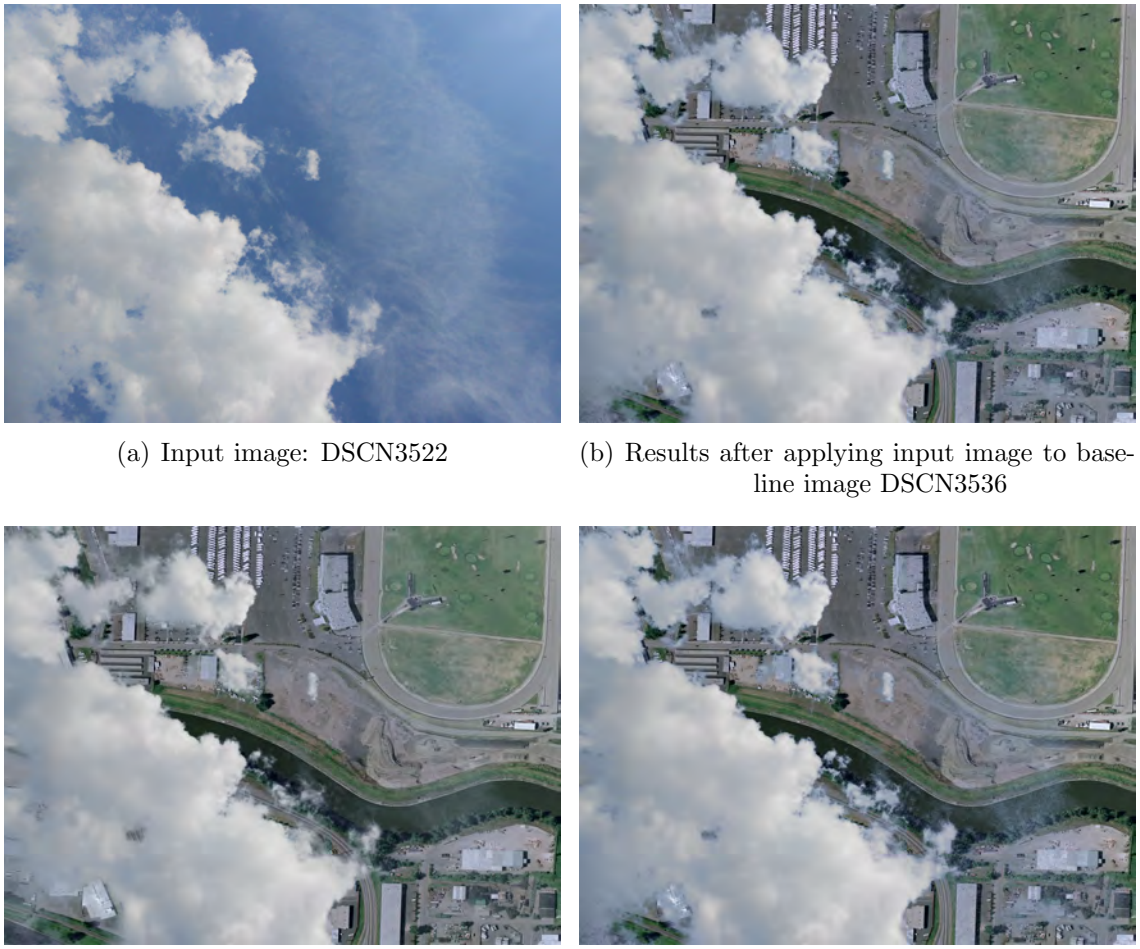
(b) Results after applying input image to baseline image DSCN3536



(c) Results after applying input image to baseline image DSCN3635

(d) Results after applying input image to baseline image DSCN3522

Figure 23. Resulting outputs for input image (a) DSCN3635 and baseline images: (b) DSCN3536; (c) DSCN3635; (d) DSCN3522.



(a) Input image: DSCN3522

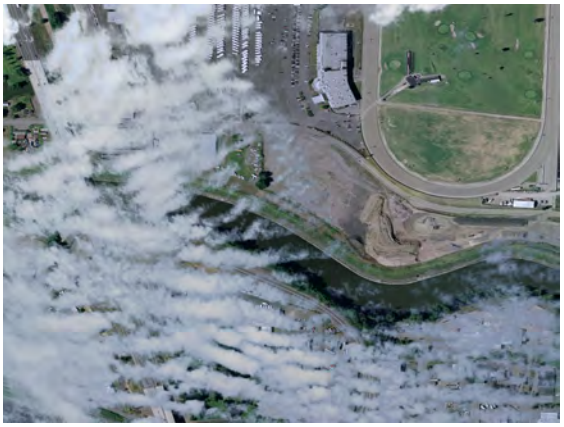
(b) Results after applying input image to baseline image DSCN3536

(c) Results after applying input image to baseline image DSCN3635

(d) Results after applying input image to baseline image DSCN3522

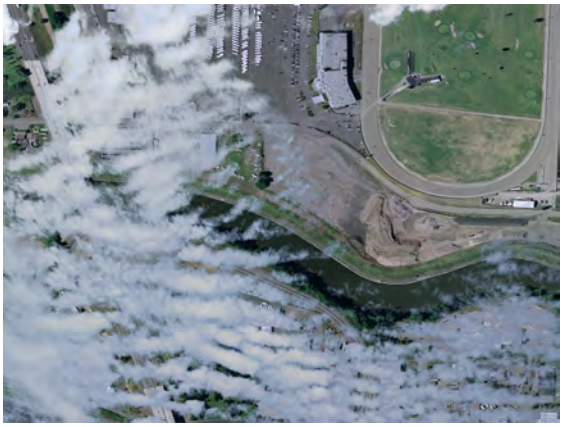
Figure 24. Resulting outputs for input image (a) DSCN3522 and baseline images: (b) DSCN3536; (c) DSCN3635; (d) DSCN3522.

(a) Input image: DSCN3548



(b) Results after applying input image to baseline image DSCN3536

(c) Results after applying input image to baseline image DSCN3635



(d) Results after applying input image to baseline image DSCN3522

Figure 25. Resulting outputs for input image (a) DSCN3548 and baseline images: (b) DSCN3536, (c) DSCN3635, (d) DSCN3522.

the blue haze around the edges: a remnant of the original sky background. Overall, however, the algorithm is able to handle the transparency calculation of images with different cloud structures. Interestingly, sometimes the algorithm produced better results when the input image was *not* the same as the baseline image. For example, the cloud overlay of Fig. 23(c) is visibly worse than the others, yet that output was specifically tailored for that image, since that was also its baseline image. This occurrence is likely due to human error. As mentioned in Section 4.1.1, four different parameters of the algorithm are adjustable. Since the "right" parameter values are partially subjective and there is no equation in which to solve for them, the algorithm is prone to some degree of human error, although small variances in parameter values do not cause significant changes to the output.

4.2 Modeling Cloud-Induced Error in Features

As mentioned in Section 3.3.1, three different error models were considered for modeling the cloud-induced error in features. The selected additive error model used for analysis was given by (18) in Section 3.3.1. This section discusses how the two rejected models were evaluated and deemed unsuitable for appropriately describing the error in features introduced by clouds.

The equations for the alternate models are given by (25) and (26):

$$\mathbf{v}_{noised} = \gamma(\alpha, \beta)\mathbf{v}_{ground} + (1 - \gamma(\alpha, \beta))\boldsymbol{\epsilon} \tag{25}$$

$$\mathbf{v}_{noised} = \gamma(\alpha, \beta)\mathbf{v}_{ground} + (1 - \gamma(\alpha, \beta))\mathbf{v}_{cloud} + \boldsymbol{\epsilon}. \tag{26}$$

Both of these equations depend on the function $\gamma(\alpha, \beta)$, which controls the trans-

parency of the entire cloud template:

$$\gamma(\alpha, \beta) = min(\alpha + \beta, 1). \tag{27}$$

In (27), α is the pixel transparency calculated using the CTCA and β is the desired transparency increase in the cloud region. $\gamma(\alpha, \beta)$ is therefore used to adjust the "cloudiness" of features. This equation increases the transparency across the entire cloud region by the amount specified by β, where β ranges from 0 to 1. The function ensures that the maximum possible transparency value of 1 is upheld by using the minimum function. When $\beta = 0$, the cloud transparency is simply the original α values; whereas when β is a maximum of 1, the entire cloud region becomes fully transparent. The result of (27) is that, while the transparency values change depending on β, the relative transparency of the pixels within the cloud regions remain the same as calculated in the CTCA (except when a maximum value of 1 has been reached). Figure 26 demonstrates the resulting transparencies of all the pixels in the image produced by $\gamma(\alpha, \beta)$ for various values of β.

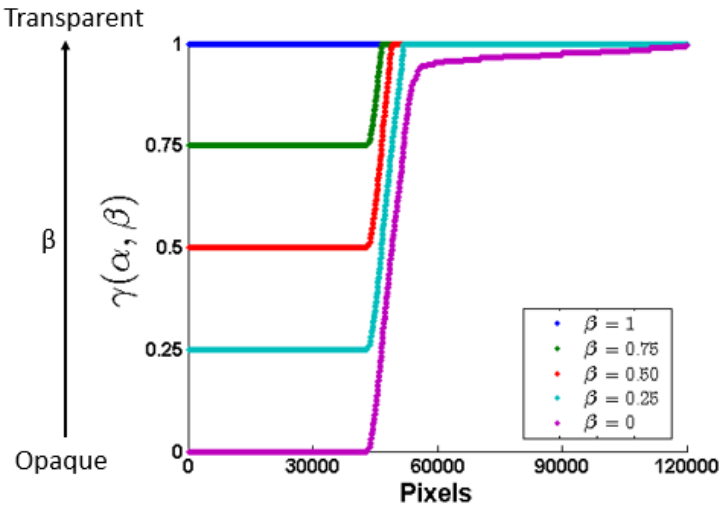


Figure 26. Adjusted cloud template transparencies using various values of β.

In (25), the noised feature is the summation of the distorted clear feature and

the varying-amplitude noise, $\boldsymbol{\epsilon}$. $\gamma(\alpha, \beta)$ gives the amount of distortion applied to $\mathbf{v}_{clear}$, while its complement specifies the amplitude of $\boldsymbol{\epsilon}$. As $\gamma(\alpha, \beta)$ decreases, higher amplitude noise is introduced into $\mathbf{v}_{noised}$. In other words, the error is modeled such that $\mathbf{v}_{noised}$ is precisely equal to $\mathbf{v}_{clear}$ when $\gamma(\alpha, \beta) = 1$ (fully transparent cloud regions) and consists only of random noise when $\gamma(\alpha, \beta) = 0$ (fully opaque cloud regions).

Equation (26) encompasses effects introduced by $\mathbf{v}_{cloud}$. This model implies that a noised feature "morphs" from a clear feature into a cloudy feature as $\gamma(\alpha, \beta)$ decreases, with the addition of random noise with constant amplitude. For example, when $\gamma(\alpha, \beta) = 1$ (fully transparent cloud regions), the noised feature is given by the clear feature with added noise. At a value of $\gamma(\alpha, \beta) = 1 - \gamma(\alpha, \beta)$, the noised feature takes on a balanced appearance of the two features, with added noise. When $\gamma(\alpha, \beta) = 0$ (fully opaque cloud regions), the noised feature takes on the full appearance of the cloud feature with the added noise.

Since both models describe the features as a function of cloud template transparency, new images are created from which to detect features. Equation (27) is applied to a single cloud template for $\beta = 0$ to $\beta = 1$, in increments of 0.01. This results in a total of 101 new images. Figure 27 shows an example of the transparency progression for $\beta = 0$, 0.25, 0.50, 0.75, and 1.

A new set of features is detected using these images. Similarly to the feature detection stage in Section 3.2.2, SURF is first used on the image without any cloud overlay to detect features on the ground. Features are then found on every other image using the same feature locations each time. 100 features on every image are collected; thus, the same feature in each of the 101 images is tracked as the cloud template transparency varies. PCA is then performed for each tracked feature positioned at a particular feature location. Figure 28 shows the PCA results for the clear, edge,

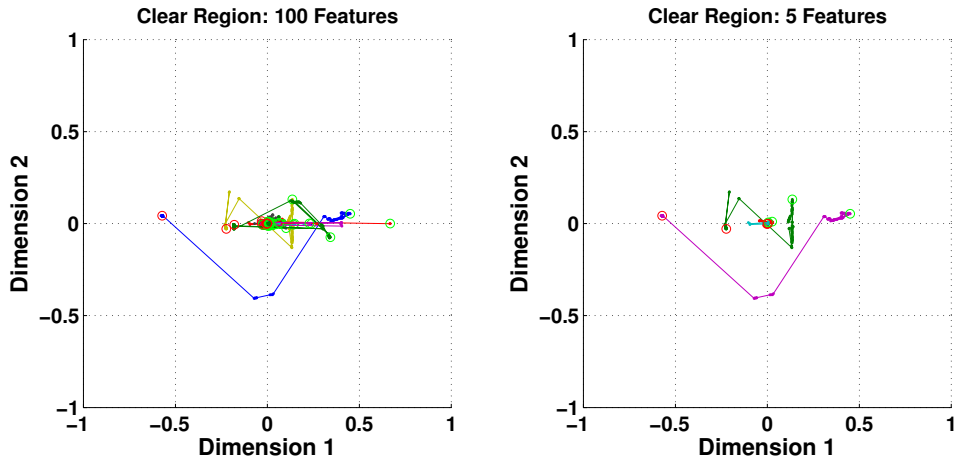
(a) $\beta = 0$ (b) $\beta = 0.25$ (c) $\beta = 0.50$

(d) $\beta = 0.75$ (e) $\beta = 1$

Figure 27. Images used to analyze the directed error models. The transparency of the cloud regions increases as β increases. The following show images using: (a) $\beta = 0$, (b) $\beta = 0.25$, (d) $\beta = 0.50$, (d) $\beta = 0.75$, and (e) $\beta = 1$.

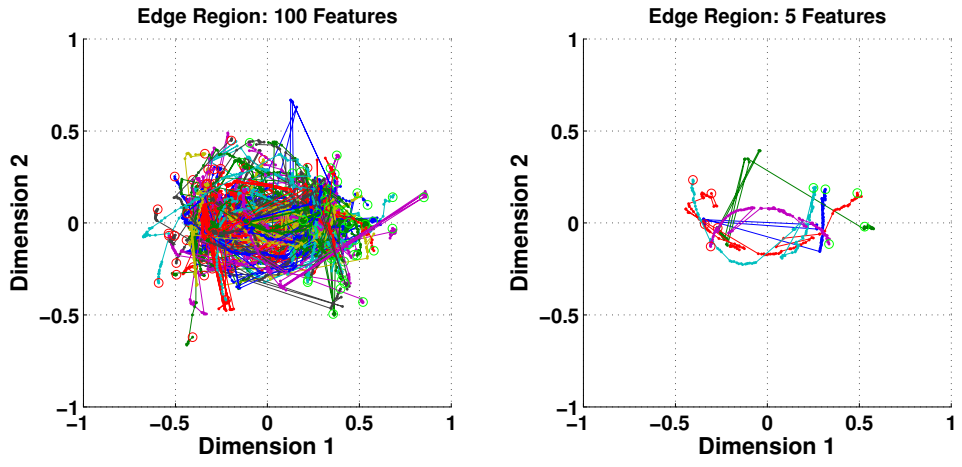
and cloud features. In each subfigure, the first dimension of the PCA data is plotted against the second dimension. This is done because, through the PCA process, the dimensions of the data are reorganized from highest to lowest amounts of variation in the data. Therefore, the first two dimensions show the maximum variation along the path of the feature vector. The data points encircled in green represent the PCA data for the default template ($\beta = 0$), while the points encircled in red represent the data with no cloud template ($\beta = 1$). Figure 29 then shows the value of the descriptors for the first dimension of the PCA data as a function of β. For ease of viewing, a selection of 5 features are plotted separately in addition to the full set of 100 features.

Figures 28(a)-28(b) display the first two dimensions of the PCA data for the clear features. The clear features have no distinct trend in movement from the green circles to the red circles. This result is not surprising, since the features in the clear regions theoretically remain exactly the same as the cloud region transparency



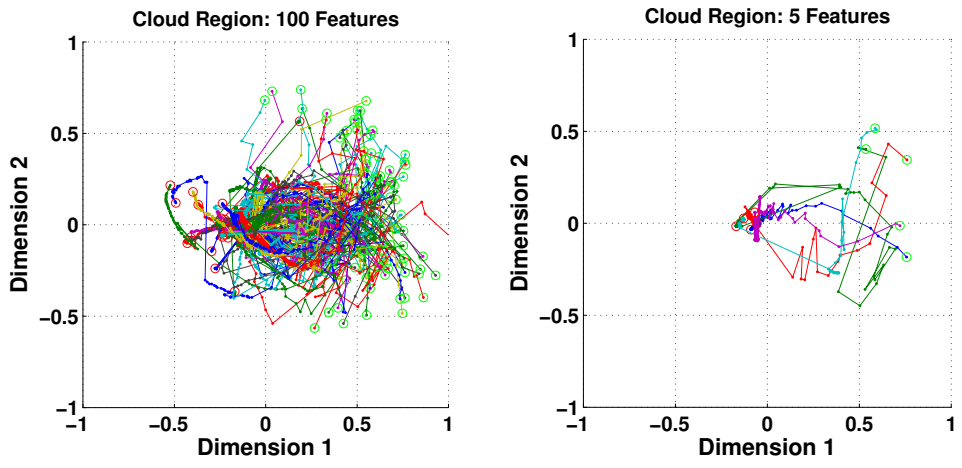
(a)

(b)



(c)

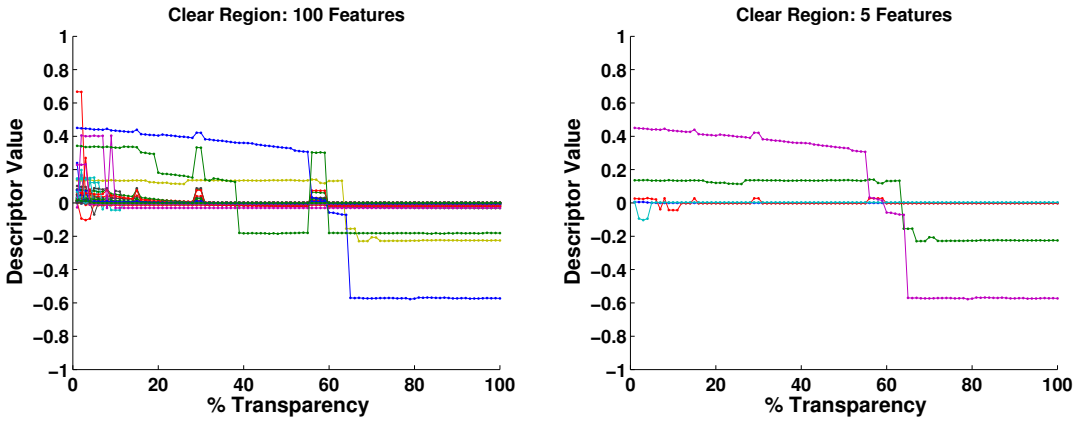
(d)



(e)

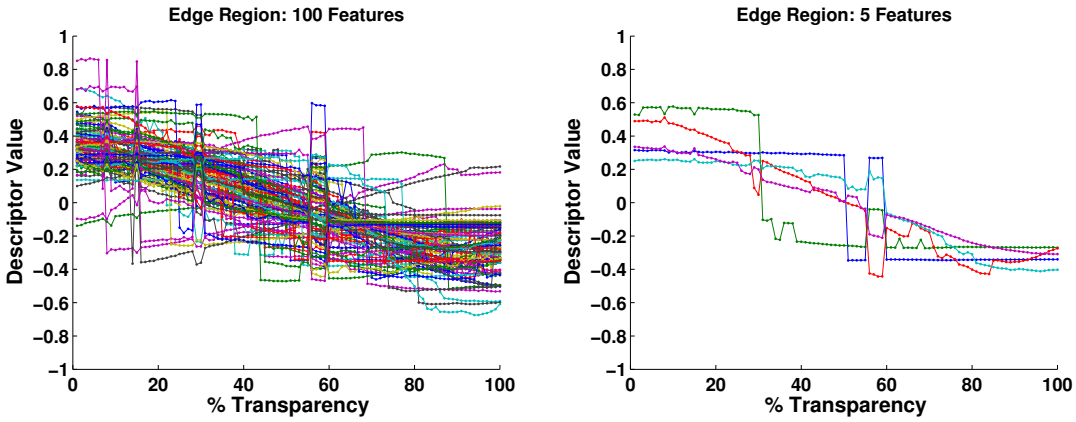
(f)

Figure 28. PCA results of the first two dimensions. Clear region: (a)-(b), edge region: (c)-(d),cloud region: (e)-(f). The plots on the left show the results for all 100 features, while plots on the right display a selection of 5 features only.



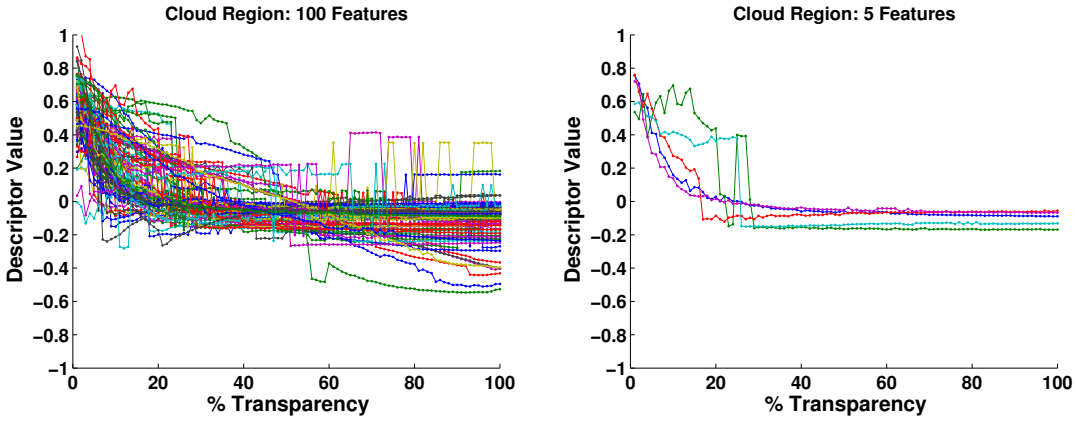
(a)

(b)



(c)

(d)



(e)

(f)

Figure 29. Descriptor values for each region plotted as a function of β. Clear region: (a)-(b), edge region: (c)-(d), cloud region: (e)-(f). The plots on the left show the results for all 100 features, while plots on the right display a selection of 5 features only.

changes. Figures 29(a)-29(b) show the change in the first dimension of the PCA descriptor values as β increases. The plot shows that, unlike what is expected, the descriptors do not stay constant as the transparency changes. There are several features that hold reasonably constant, but many also jump in value unexpectedly. It is therefore not reasonable to model the ground features as a function of transparency.

Figures 28(c)-28(d) show the PCA results for the edge features. In theory, the features in the edge regions are the top contenders for showing directed error because the features in this region "see" the ground to some extent for all values of β. However, looking at the figures, the movement from the starting green circle to the ending red circle varies widely from feature to feature. The lack of consistent trends as the features progress into full transparency is also apparent in Figs. 29(c)-29(d). The descriptor values generally form an inverse s-curve as β increases, but within each feature, multiple sharp increases and decreases in the values are common. Thus, the data show that the edge features cannot reliably be described as a function of the transparency.

Lastly, Figs. 28(e)-28(f) display the results for the cloud features. Looking at the figures, the results for the cloud region have the most consistency from start to end for each particular feature, but the variation from feature to feature still widely varies. The green circles for each feature tend to begin near the right side of the plot and progress to the left side. However, the path in which they take from either side of the plot changes significantly between features, so no generalization of the movement can be made. The cloud features also show modest support for modeling features as a function of the cloud transparency. Figures 29(e)-29(f) show that the descriptor values tend high in the presence of clouds ($\beta = 0$), sharply decrease in value, and then settle on a constant value at around a value of $\beta = 0.25$. However, even in this scenario, the trend is not strong enough to rely upon. As seen in Fig. 29(f), a few

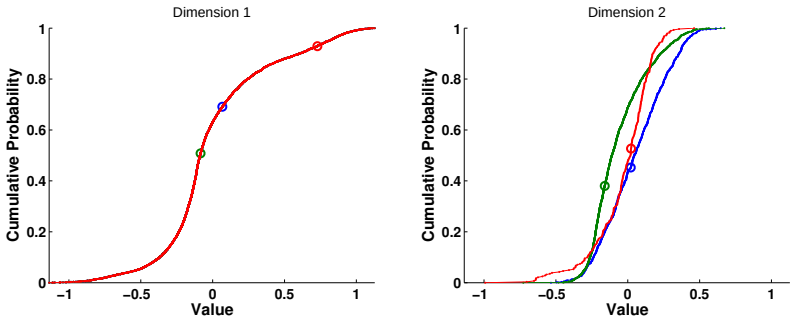
features follow the trend, but the descriptor values of the features that do not are offset by as much as 0.4.

In conclusion, the results do not support either method of modeling the features as a function of their transparency. The analysis failed to show that the undirected and directed error models would simplify the error model. Therefore, these methods are rejected and the basic additive error model given in 18 is chosen.

4.3 Synthetic Dataset Creation

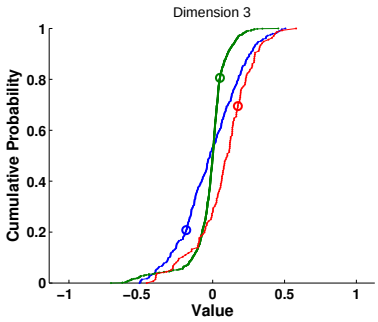
For each feature category, a synthetic dataset consisting of $1x10^6$ 8-dimensional vectors is generated using the random sampling approach described in Section 3.3.3. The synthetic datasets are designed to be statistically similar to the error vectors in the PCA space. A demonstration of Algorithm 1 applied 3 times to the error vectors in the edge region is given in Fig. 30.

Each color in the plots represents the progression through the dimensions for a single application of the algorithm. The y-values of the circles represent the uniformly-sampled random numbers that give a cumulative probability, while the x-values correspond to the interpolated outputs that give the value of a synthetic data point associated with that probability. The changes in the curves in each plot show that the data are not independent. This implies that only local data must be used, as opposed to the full set of data. As seen in the figure, the number of data points quickly decreases through the progression of the dimensions if the random number falls on the outliers of the data, as occurs in the blue and red runs. However, the random numbers generated for the green run all happen to fall around the mean of the data at 0. Since most of the data is concentrated around 0, the green run retains a significant amount of accepted indices. The special case handling is apparent for the blue run in Fig. 32(h). Only two data points are present, so the algorithm randomly

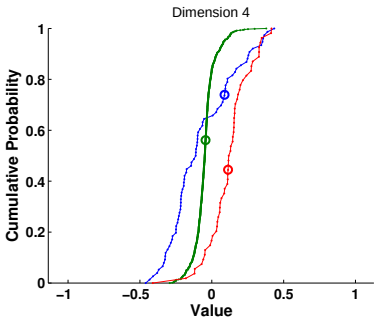


(a) Dimension 1

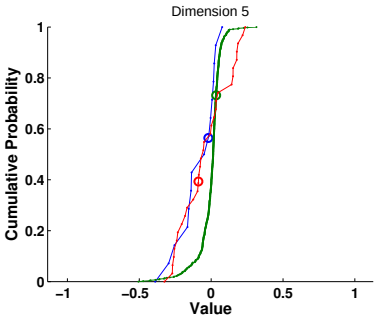
(b) Dimension 2



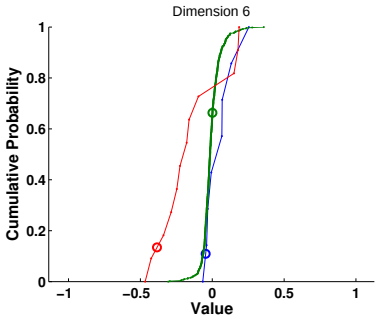
(c) Dimension 3



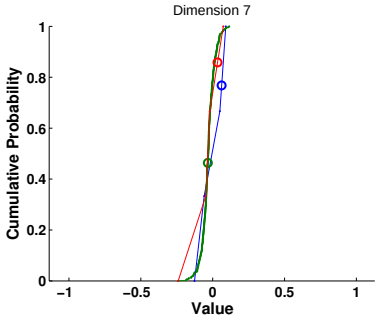
(d) Dimension 4



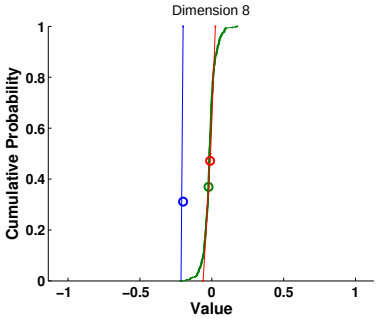
(e) Dimension 5



(f) Dimension 6



(g) Dimension 7



(h) Dimension 8

Figure 30. Example of the synthetic data generation process described in Algorithm 1, applied to the error vectors in the edge region.

a selected one of the two points as its output.

After repeating the synthetic data generation process to obtain a dataset consisting of $1x10^6$ observations, the synthetic dataset is evaluated. The following discussion provides the resulting PDFs of the synthetic datasets and their comparison to those of the error vectors, as well as a count of the number of times that special handling was required for the interpolation step in the algorithm. Recall that special handling was required any time the interpolation step of Algorithm 1 was faced with 3 or less points between which to interpolate. This was broken up into two cases. If 0 data points were present, the output of the interpolation was set to 0. Otherwise, if 1 to 3 points were present, then one of the points was randomly selected as the output.

Figures 31 through 33 display the comparisons of the PDFs between the PCA and synthetic datasets for the clear, edge, and cloud errors, respectively. In each figure, the blue circles represent the PDFs for the error vectors in the PCA space, while the red stars represent the synthetic dataset. Table 3 gives the frequency with which special case handling was required. Additionally, Tables 4 through 6 give the statistics on the similarity of the synthetic dataset to that of the original dataset. They describe the mean and standard deviation of both the original and synthetic datasets, as well as the percent differences in the standard deviations. The percent differences are not calculated for the means, since the mean of the PCA dataset is essentially 0.

Table 3. Statistics for Special Case Handling

Category	Random Selection %	Assign Zero %
Clear	0.02	0.18
Edge	0.81	12.70
Ground	0.85	14.36

Figure 31 shows the comparison of the transformed dataset to the synthetic dataset for the clear features. The synthetic dataset is nearly identical for each of the 8

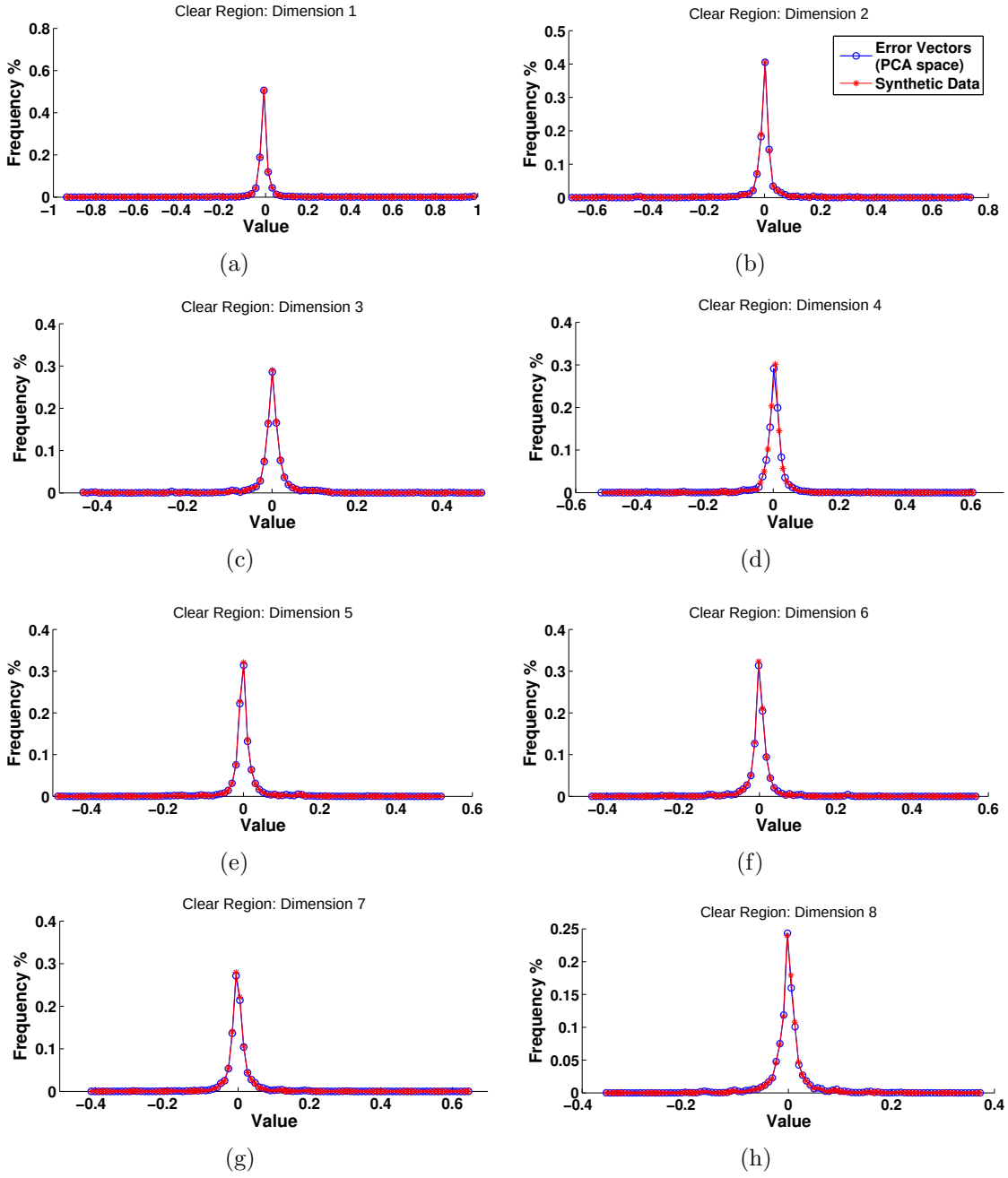


Figure 31. Histogram comparison for the transformed error vectors and the synthetic dataset for the clear feature region. Plots (a)-(h) show the results for dimensions 1-8, respectively.

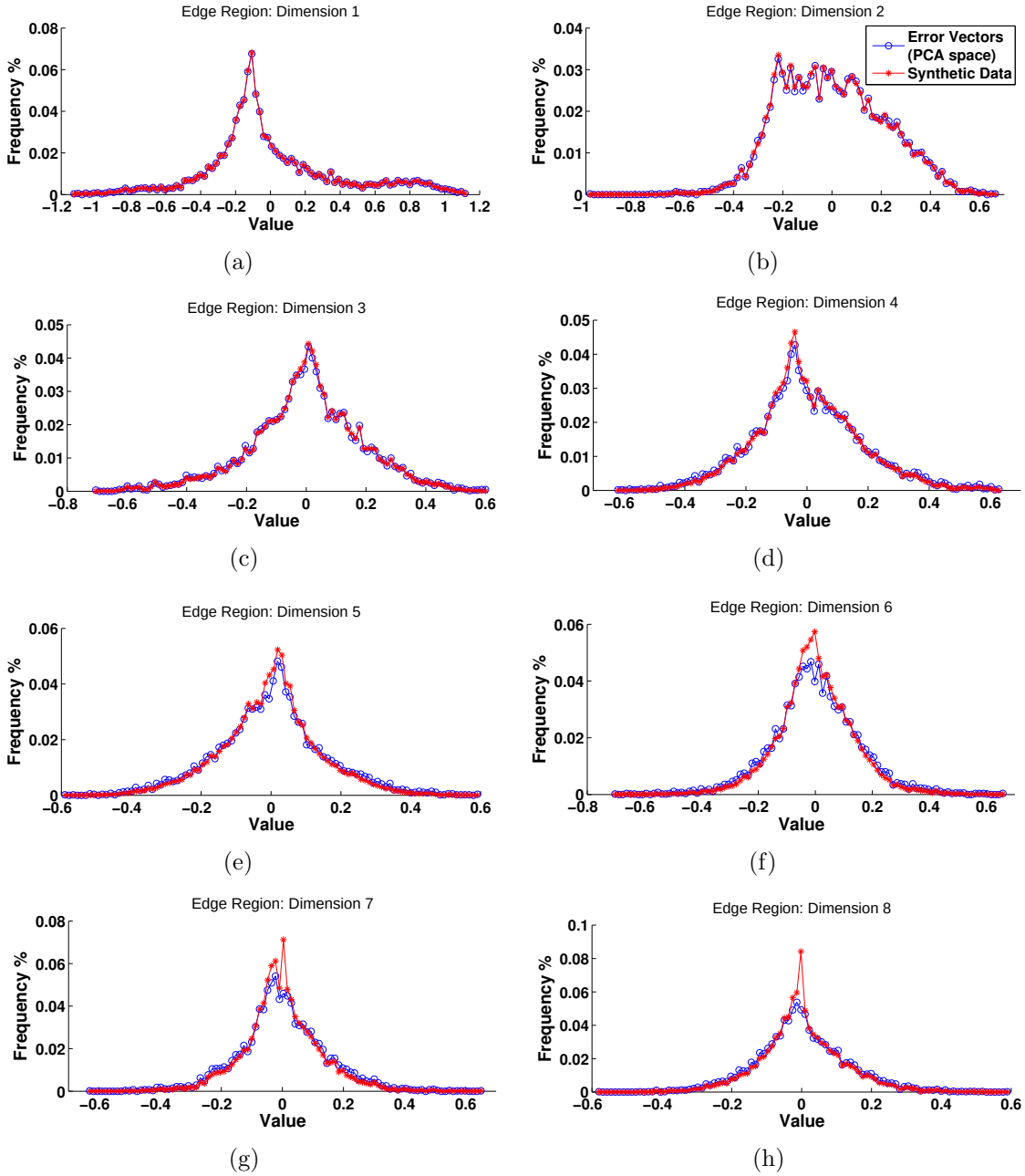


Figure 32. Histogram comparison for the transformed error vectors and the synthetic dataset for the edge feature region. Plots (a)-(h) show the results for dimensions 1-8, respectively.

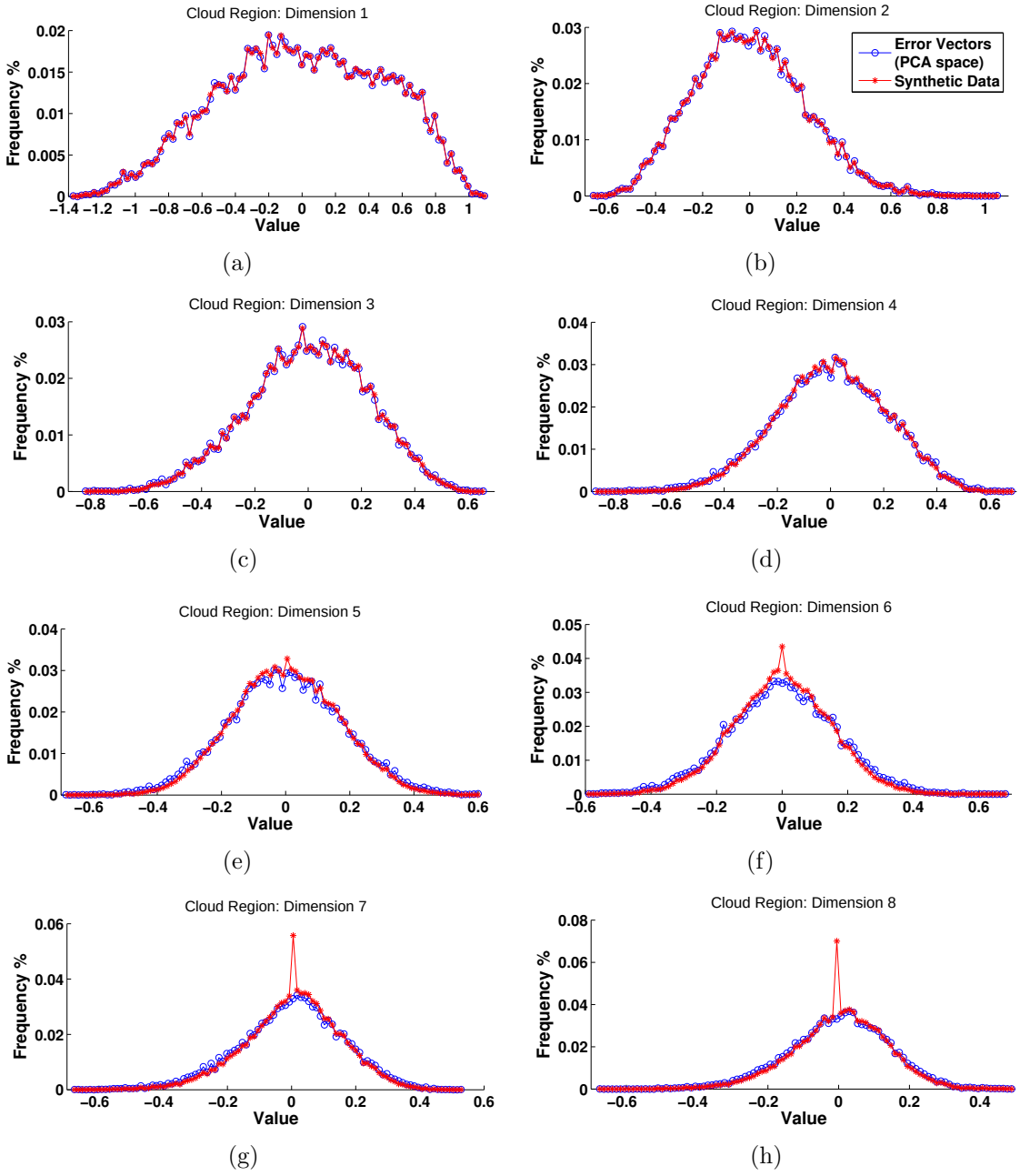


Figure 33. Histogram comparison for the transformed error vectors and the synthetic dataset for the cloud feature region. Plots (a)-(h) show the results for dimensions 1-8, respectively.

Table 4. Statistics for Synthetic Dataset: Clear Region

Data Dimension	Mean: Transformed	Mean: Synthetic	Std. Dev: Transformed	Std. Dev: Synthetic	Percent Difference
1	0.000	0.000	0.106	0.106	0.000
2	0.000	0.000	0.084	0.082	2.23
3	0.000	0.001	0.059	0.054	8.47
4	0.000	0.001	0.054	0.045	16.7
5	0.000	0.000	0.047	0.041	12.8
6	0.000	0.000	0.044	0.038	13.6
7	0.000	-0.001	0.042	0.035	16.7
8	0.000	0.000	0.038	0.033	13.2

Table 5. Statistics for Synthetic Dataset: Edge Region

Data Dimension	Mean: Transformed	Mean: Synthetic	Std. Dev: Transformed	Std. Dev: Synthetic	Percent Difference
1	0.000	0.000	0.375	0.375	0.000
2	0.000	-0.002	0.211	0.210	0.474
3	0.000	0.001	0.188	0.183	2.66
4	0.000	0.000	0.179	0.168	6.15
5	0.000	-0.002	0.155	0.139	10.3
6	0.000	0.000	0.151	0.133	11.9
7	0.000	-0.002	0.139	0.124	10.8
8	0.000	0.000	0.129	0.117	9.30

Table 6. Statistics for Synthetic Dataset: Cloud Region

Data Dimension	Mean: Transformed	Mean: Synthetic	Std. Dev: Transformed	Std. Dev: Synthetic	Percent Difference
1	0.000	0.001	0.485	0.485	0.000
2	0.000	-0.001	0.235	0.234	0.426
3	0.000	0.002	0.223	0.222	0.448
4	0.000	0.003	0.213	0.206	3.29
5	0.000	0.001	0.177	0.164	7.34
6	0.000	-0.001	0.164	0.147	10.4
7	0.000	0.005	0.160	0.148	7.50
8	0.000	0.003	0.141	0.131	7.09

dimensions. As seen in Table 4, the standard deviation of the transformed dataset increases with the dimensions, as expected due to the PCA process. The synthetic dataset matches the standard deviation closely, but not perfectly. The difference

between the two standard deviations is at most 16.20% for each dimension, with an average percent difference of 10.48%. The mean of the synthetic dataset is not always 0, although still sufficiently close to 0. Referring to Table 3, the frequency with which either of the special case handling was required is minimal, and is not noticeable in the PDFs of the data.

Figure 32 displays the PDF comparisons for the edge features. The form of the PDFs for the transformed edge error vectors is more complex than those of the transformed clear error vectors. There are more irregularities in the data, such as the spikes and dips in Fig. 32(b). The PDFs of the synthetic data still closely match the transformed PDFs, however, the special case handling is evident in Figs. 32(f) through 32(h). For the sixth through eigth dimensions of the data, the number of values equal to 0 within the synthetic dataset is up to approximately 30% higher as compared to the transformed error vectors. As seen in Table 3, the number of times that a random selection of the inputs was assigned to the output is less than 1%, but the frequency with which outputs are assigned to 0 is much larger, at nearly 13%. Lastly, Table 5 shows that the mean of the synthetic dataset for each dimension is near 0, but not as close as the transformed error vectors. The percent differences in the standard deviations are much smaller than those for the clear region with a maximum percent difference of 12.25% and an average of 6.50%.

The last set of comparisons can be seen in Fig. 33. These plots show the behavior of the transformed error vectors and the synthetic dataset for the cloud region. Much like with the edge region, all of the irregularities in the transformed data are reflected in the synthetic data. Also similarly to the edge region is the clear increase in the number of times a value of 0 occurs for the synthetic dataset. Table 3 shows that the frequency of special case handling is slightly higher than for the edge region. The number of times a random input was assigned to the output is still largely insignificant

at only 0.85%, but the output is forced to take on a value of zero over 14% of the time. However, even with the increased percentage of the special case handling, Table 6 shows that the percent differences for the standard deviations in the cloud regions are smaller than those for the edge regions. The maximum difference is roughly 10%, with an average difference of 4.65% between all of the dimensions. The means of the synthetic dataset are once again near 0, closely matching the mean of the transformed data.

Thus, in conclusion, the synthetic dataset can be described as statistically similar to the transformed error vectors on which it is based. The synthetic data are more biased towards 0 due to the nature of the interpolation process, but otherwise remain true to the behavior of the transformed error vectors. Therefore, the use of the synthetic dataset to create the synthetic errors is justified and can be assumed to produce similar results that the transformed error vectors would. The advantage of using the synthetic dataset is that it consists of many more observations than the transformed error vectors, which leads to more generalized results.

4.4 Data Metrics

To verify the validity of the synthetic dataset of synthetic error vectors, a metric is developed to compare it to the original dataset of error vectors directly derived from the images. The metric helps determine if the synthetic dataset is statistically similar to the original. The metric combines two approaches to verify similarity of the synthetic error vectors to the original error vectors. For purposes of comparison, a dataset of Gaussian noise is generated and compared to the original, as well. The Gaussian noise is designed such that each dimension is transformed into the same PCA-space as the synthetic dataset, scaled by the weights generated through the PCA process, and offset by the appropriate mean values. Equation (28) mathematically

demonstrates how the Gaussian dataset $\boldsymbol{\epsilon}_\omega$ is created:

$$\boldsymbol{\epsilon}_\omega = \boldsymbol{\Omega}^T \mathbf{G} \mathbf{C}^T + \boldsymbol{\mu}, \tag{28}$$

where $\boldsymbol{\Omega}$ is a 64x(1x10^6) matrix of Gaussian random noise, $\mathbf{G}$ is a 64x64 diagonal matrix consisting the of scaling coefficients, $\mathbf{C}$ is the Mx64 matrix that transforms the data to the PCA space, and $\boldsymbol{\mu}$ is the $1x64$ vector of means added on to each dimension.

4.4.1 Metric Creation.

The first part of the metric checks to see if the dataset in question has similar values to the original dataset $\underline{\underline{\boldsymbol{\epsilon}}}_{orig}$. This approach counts the number of instances in which the distance between a single query error vector $\boldsymbol{\epsilon}_{query}$ and each of the original error vectors is less than a specified threshold, t. The number of elements in a vector is designated with $\#(\cdot)$. The final results are normalized by the number of error vectors in the original set of error vectors, M. If a query dataset has similar values to the original, the histograms of this approach should approximately match. The implementation of this approach is shown in (29):

$$h_1(\boldsymbol{\epsilon}_{query}) = \frac{\#(\|\boldsymbol{\epsilon}_{query} - \underline{\underline{\boldsymbol{\epsilon}}}_{orig}\| < t)}{M}. \tag{29}$$

The second part verifies that the query error vectors have similar values to their own datasets, $\underline{\underline{\boldsymbol{\epsilon}}}_{query}$. This approach counts the number of instances in which the distance between a single query error vector and each of the query error vectors is less than the same threshold t as in (29). As with the first approach, the final results are normalized by $\#(\boldsymbol{\epsilon})$. In essence, if the query dataset has a similar distribution of values to the original, the histograms of their results should be similar. This approach

is implemented using (30).

$$h_2(\boldsymbol{\epsilon}_{query}) = \frac{\#(\|\boldsymbol{\epsilon}_{query} - \underline{\boldsymbol{\epsilon}}_{query}\| < t)}{\#(\boldsymbol{\epsilon}_{query})}. \tag{30}$$

Lastly, both approaches are combined to form the metric used to evaluate the query datasets. The metric simply takes the magnitude of the differences in (29) and (30):

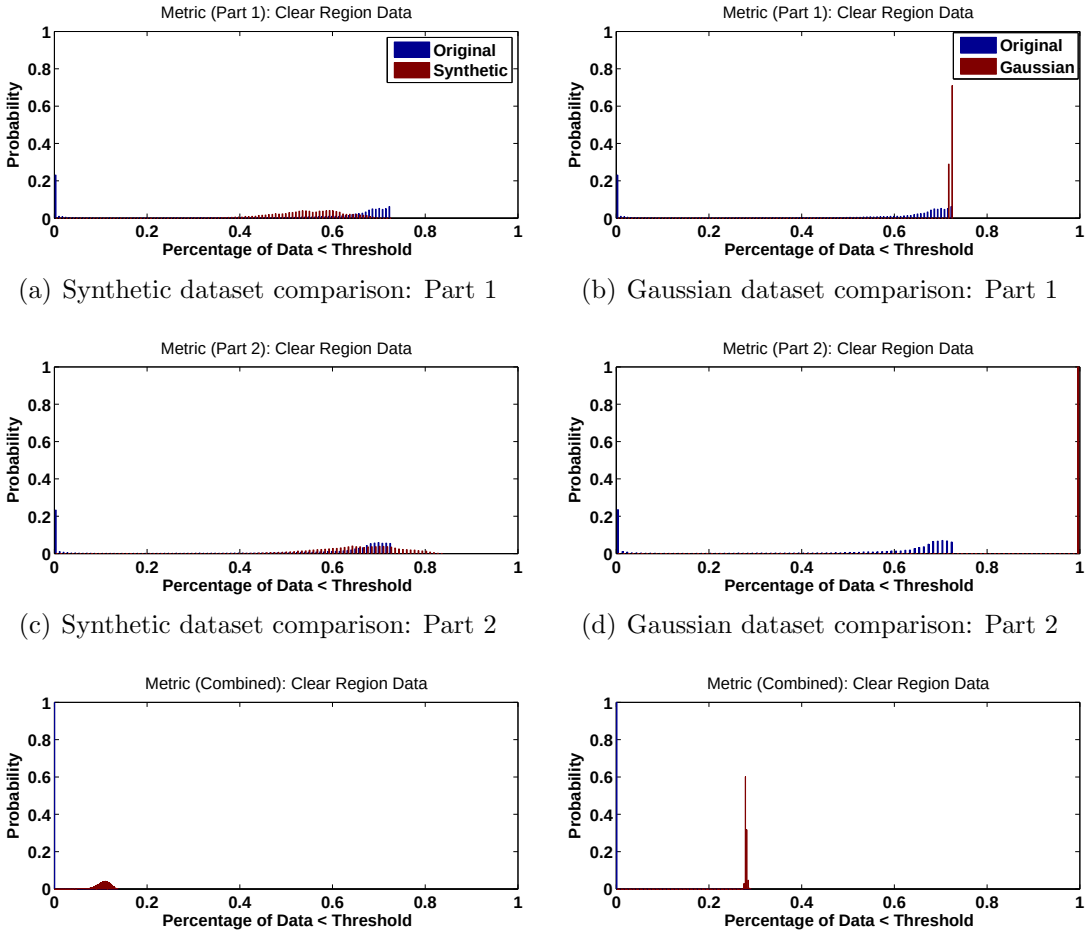
$$h = |h_1 - h_2|. \tag{31}$$

The query datasets consist of the original, synthetic, and Gaussian error vectors. Each one is evaluated separately, then the results of the synthetic and Gaussian sets are compared to those of the original.

4.4.2 Metric Evaluation.

For each of the three feature regions, the threshold t is set to 0.1, the same threshold applied when creating the synthetic dataset. The results are displayed in Figs. 34 through 36.

The evaluation of the synthetic dataset for the clear region is given in Fig. 34. Comparing Part 1 of the metric for the synthetic and Gaussian datasets, one can see that the distribution of the synthetic data matches that of the original data significantly more than the Gaussian data. Part 2 of the metric shows that the Gaussian data is much more likely to have data that are similar to itself than either the original data or the synthetic data. The combined metric then reveals that the synthetic dataset matches the distribution of the original data more so than the Gaussian dataset. The mean of the synthetic data is approximately 0.1, while the mean of the Gaussian dataset is slightly less than 0.3. Thus, the synthetic data for the clear region matches the distribution of the original data more than the Gaussian



(a) Synthetic dataset comparison: Part 1

(b) Gaussian dataset comparison: Part 1

(c) Synthetic dataset comparison: Part 2

(d) Gaussian dataset comparison: Part 2

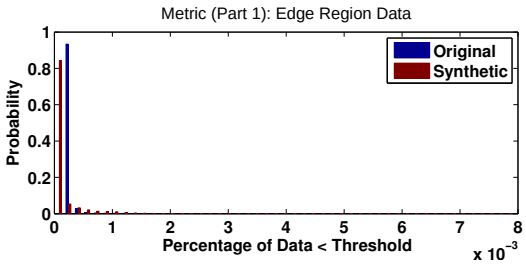
(e) Synthetic dataset comparison: Combined

(f) Gaussian dataset comparison: Combined

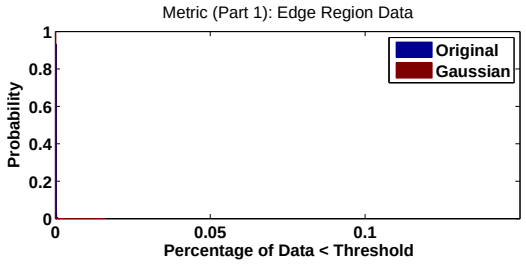
Figure 34. Results of the metric calculations for the clear region using a threshold of $t = 0.1$. The left column shows the results for the synthetic data, while the right column shows results for the Gaussian random data.

random data.

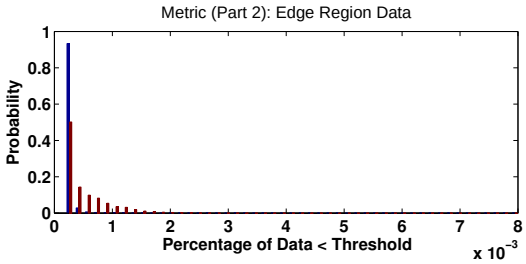
Fig. 35 displays the results for the data in the edge region. From Part 1 of the metric for the synthetic and Gaussian datasets, one can see that both distributions are similar to the original data. In both cases, there is a high probability that only small amounts of data are less than the given threshold. Part 2 of the metric shows that, similarly to the original dataset, there is a high probability that the synthetic dataset does not have values similar to itself. However, there is a higher probability that the Gaussian data contains more values similar to itself. Lastly, the combined



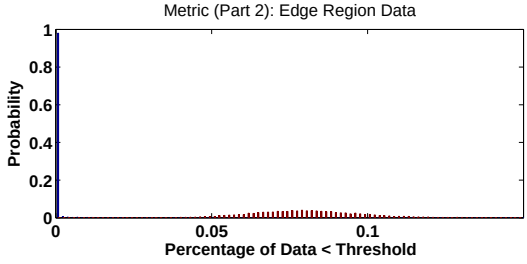
(a) Synthetic dataset comparison: Part 1



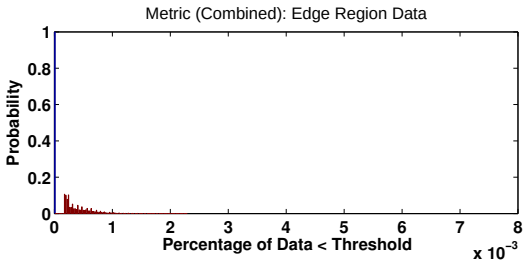
(b) Gaussian dataset comparison: Part 1



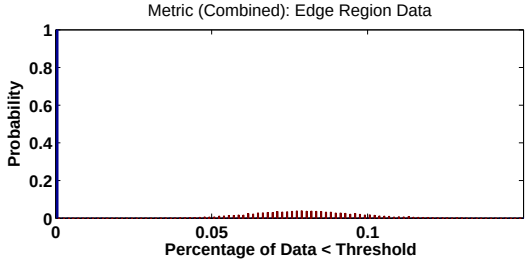
(c) Synthetic dataset comparison: Part 2



(d) Gaussian dataset comparison: Part 2



(e) Synthetic dataset comparison: Combined

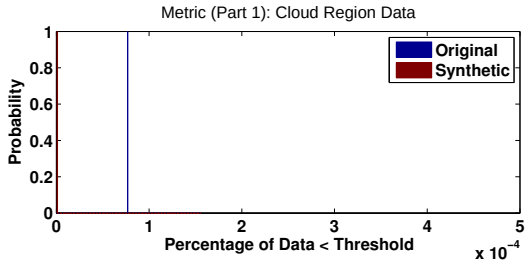


(f) Gaussian dataset comparison: Combined

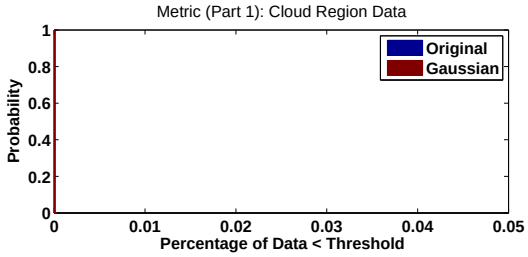
Figure 35. Results of the metric calculations for the edge region using a threshold of $t = 0.1$. The left column shows the results for the synthetic data, while the right column shows results for the Gaussian random data.

metric shows that the synthetic dataset matches the distribution of the original data more than the Gaussian dataset. The mean of the synthetic data is approximately $0.25\text{x}10^{-3}$, whereas the mean of the Gaussian dataset is significantly higher at 0.75. The metric therefore shows that the synthetic data corresponding to edge regions has more similarity to the original dataset than the Gaussian random data.

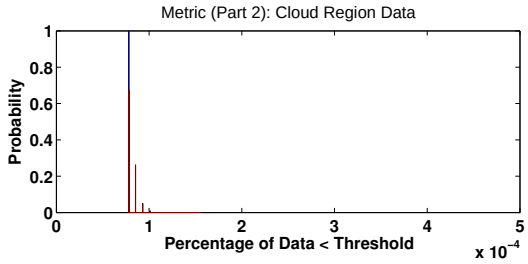
The cloud region comparisons are shown in Fig. 36. Part 1 shows that the Gaussian data contain more similarity to the distribution of the original dataset. However, the histograms in Fig. 36(a) and 36(b) shows that the results of Part 1 were all 0, which



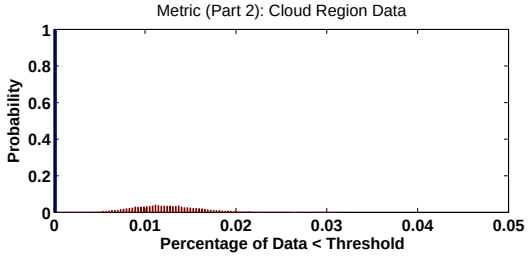
(a) Synthetic dataset comparison: Part 1



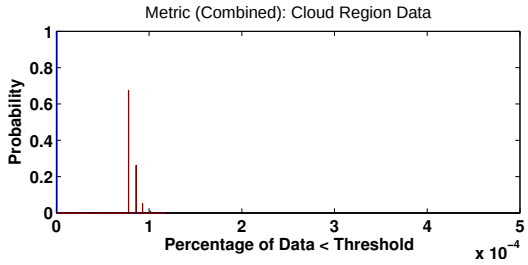
(b) Gaussian dataset comparison: Part 1



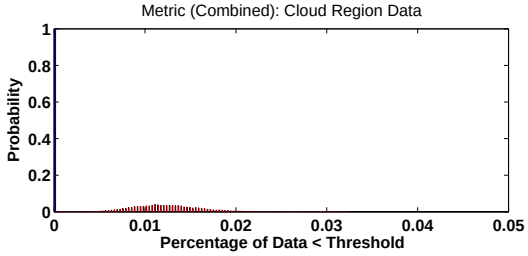
(c) Synthetic dataset comparison: Part 2



(d) Gaussian dataset comparison: Part 2



(e) Synthetic dataset comparison: Combined



(f) Gaussian dataset comparison: Combined

Figure 36. Results of the metric calculations for the cloud region using a threshold of $t = 0.1$. The left column shows the results for the synthetic data, while the right column shows results for the Gaussian random data.

implies that the selected threshold was too small. A larger threshold may reveal different results. Similarly to the clear and edge regions, Part 2 of the metric shows that the Gaussian data is much more likely to have data that are similar to itself than either the original data or the synthetic data. The combined metric reveals that the synthetic dataset matches the distribution of the original data more so than the Gaussian dataset. The histogram is centered at approximately $0.75\text{x}10^{-3}$ for the synthetic data, and 0.012 for the Gaussian data. The synthetic data for the clear region therefore matches the distribution of the original data more than the Gaussian

random data.

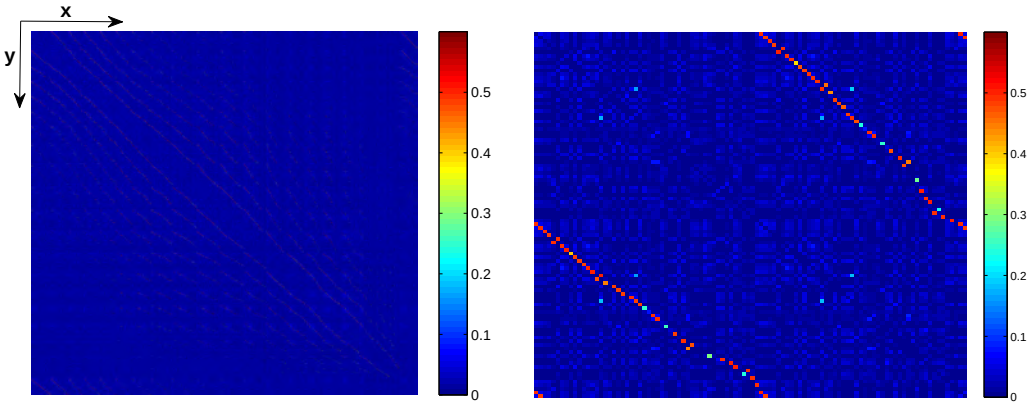
In conclusion, the synthetic data for each of the regions is shown to perform better than the Gaussian random data. Although the metric provides only a qualitative comparison of the similarity of the data, it gives insight that the synthetic dataset shares more similarity of data to the original dataset than a random dataset does.

4.5 Monte-Carlo Simulations

The final steps of the research process are the Monte-Carlo simulations used for empirically determining the probability of a mismatch between features. Recall from Section 3.3.5 that a mismatch for feature a is defined as the instance when feature b is incorrectly identified as feature a. The probability of a mismatch was given by (24). This equation is used in the Monte-Carlo simulations, where 1,000 features are compared amongst each other. The selected features are the truth features derived from the original set of truth images in Section 3.2.2. Additionally, for each feature pair, 10,000 synthetic error vectors are randomly selected from the pool of $1\text{x}10^6$ total synthetic error vectors. As with before, the calculation of the probability of a mismatch is segmented into the clear, edge, and cloud features.

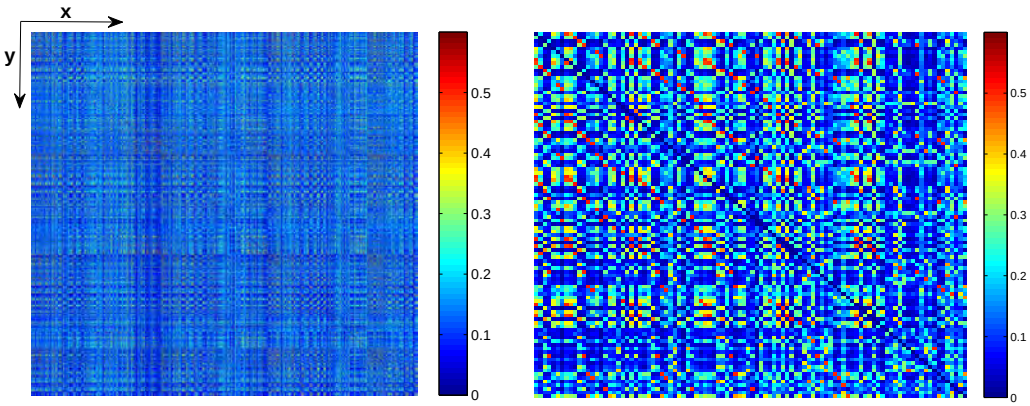
Figures 37 through 39 give the results of the simulations for the clear, edge, and cloud regions, respectively. To see the results more clearly, each result is accompanied by a subset of the first 100x100 feature comparisons. Going along the y-axis of the figures represents the probability of a mismatch for feature a, while the x-axis gives the probability of a mismatch for feature b.

By looking at the figures, one can see near symmetry about the axis running from the top left corner to the bottom right corner. This implies that, although not identical, the probability of a mismatch between feature a and b is similar to the probability of a mismatch between feature b and a.



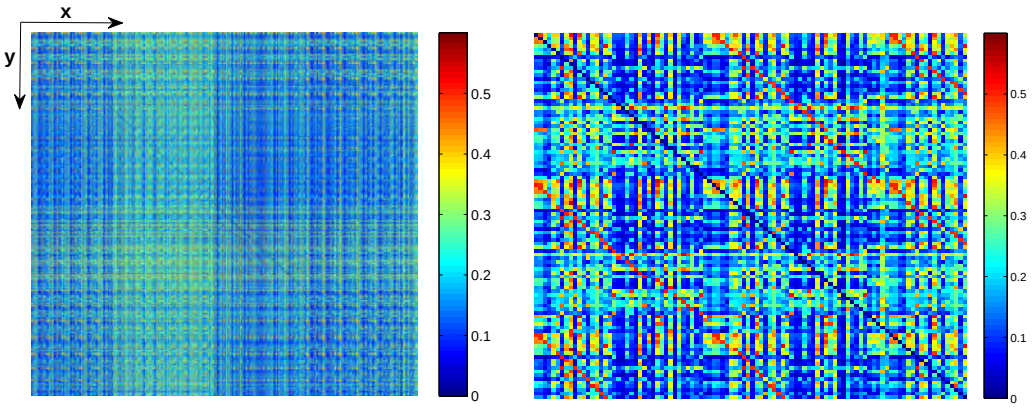
(a) Full results (b) First 100x100 results

Figure 37. Probability of mismatch between: (a) all 1000 clear features, (b) first 100 clear features.



(a) Full results (b) First 100x100 results

Figure 38. Probability of mismatch between: (a) all 1000 edge features, (b) first 100 edge features.



(a) Full results (b) First 100x100 results

Figure 39. Probability of mismatch between: (a) all 1000 cloud features, (b) first 100 cloud features.

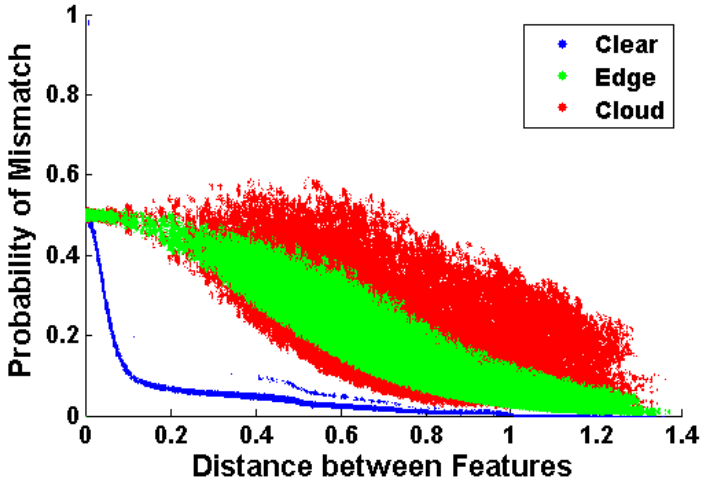


Figure 40. Probability of a mismatch plotted against the distance between feature pairs for each feature region.

Figure 40 plots the probability of a mismatch for each feature region as a function of the distance between each pair of compared features. The distance is measured as the norm of the differences in each 64-dimensional feature descriptor. For features located in the clear region, the probability of a mismatch sharply declines as the distance increases. A small cluster of features produce a probability of nearly 1, although this is likely due to the randomness of the process involved. Other than that abnormality, the maximum probability of a mismatch is 51%, which drops down

to approximately 10% at a distance of 0.1. Looking at the edge region, the results show a gradual decrease in the likelihood of mismatching features. The smallest distance at which the probability of a mismatch for edge features falls below 10% occurs at a distance of 0.68, while the largest distance is 1.1. Lastly, the results for the cloud features are the poorest, and also contain the most variation. The maximum mismatch probability for cloud features is 60%, which is not drastically different from the other regions. However, the probability of a mismatch is much less defined as a function of the distance between feature pairs. The distances for which the probability of a mismatch is 10% for the cloud features range from 0.59 to 1.33, a difference of 0.74.

Overall, the results of Fig. 40 show that for clear features, if the distance between features is known, then the probability of those features mismatching is precisely known. The probability of edge features is only known to approximately 20% in the worst-case scenario. Lastly, the distance between cloud features is not able to accurately predict the probability of mismatching features. However, this figure provides valuable insight as to the likelihood of being able to determine the probability of mismatching features based on the regions in which they are located.

Histograms of the probability of a mismatch for each feature regions are shown in Fig. 41. The plot reflects that 70% of the clear features have 0-0.3% likelihood of mismatching. In fact, approximately 90% of the clear features have less than a 10% likelihood of being mismatched. The probability of a mismatch for the edge and cloud features are similar, but significantly different than the clear features. The mean probability is slightly smaller for edge regions at a value of approximately 5%, while the average probability for the cloud region is 10%.

In summary, the probability of a mismatch between features is successfully determined for each feature region. It is found that mismatches between features in the

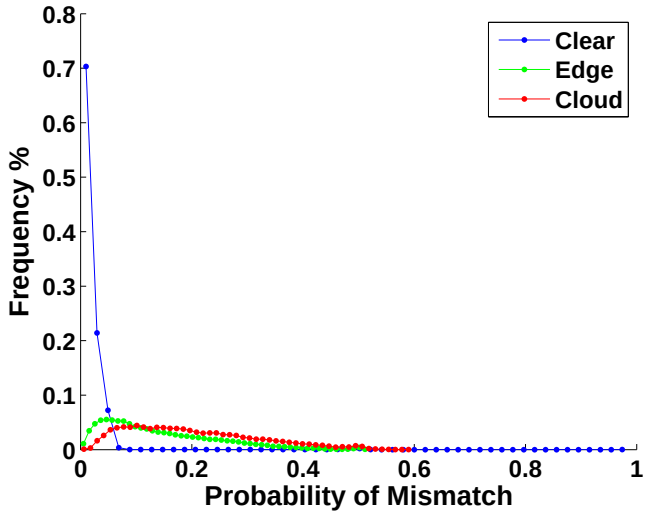


Figure 41. Histogram of the probability of a mismatch for each feature region.

clear region are highly unlikely to mismatch, which matches expectations. Features in the edge region are significantly more likely to be mismatched, but on average, are only mismatched about 5% of the time. The highest likelihood of mismatched features occurs in the cloud region, at an average occurrence of 10%.

4.6 Summary

This chapter described and analyzed the results produced. Section 4.1 showed the performance of the CTCA when applied to images containing various cloud structures. Section 4.2 detailed the two rejected methods for modeling the cloud-induced error in features. A comparison of the synthetic dataset to the PCA dataset from which it is based was made in Section 4.3. A description and evaluation of the metric developed to measure the validity of the synthetic error vectors to the original dataset of error vectors was described in Section 4.4. Lastly, results of the Monte-Carlo simulations for determining the probability of a mismatch between features were given in Section 4.5. Final conclusions about the research and a discussion of potential future work is described in Chapter V.

V. Conclusions & Future Work

The following chapter summarizes the conclusions made from the findings of the research and provides suggestions for future work into the research area.

5.1 Conclusions

The research was developed through three main stages. First, an algorithm was created that calculates the transparency of cloud images at the pixel level. Calculating the transparency allows the algorithm to then overlay the cloud image onto any ground-view image to create realistic cloudy aerial images. This algorithm was used to simulate a motion sequence of an aircraft flying in cloudy conditions. The second stage used the cloud-covered images to extract features in the three distinct regions of the image: areas without any clouds, areas along the edges of the clouds, and areas fully immersed in clouds. Performing separate analyses in each of the three regions allows the error in features due to clouds to be more extensively studied. Lastly, a method was developed in which to evaluate the error introduced into features by clouds in terms of the likelihood of two features being mismatched.

The cloud transparency algorithm is shown to perform effectively for all cloud images tested. Although the research uses only a single baseline image and a single input image to produce the cloud template used for the cloud overlays, the tests show that other images perform with similar effectiveness. The algorithm successfully calculates realistic transparencies for all combinations of baseline images and input images. While more images with varying cloud structures and range of color content would need to be tested to generalize results for the algorithm, the algorithm is undoubtedly capable of producing realistic cloud transparencies with the quality necessary for the research.

The method with which features were detected on the cloud overlay images was carefully developed, though suffers from the limitations on the forced matching process. However, separation of analysis into the three cloud regions allowed a close investigation into the effects of clouds on the description of features.

The method for calculating the probability of a mismatch is ideal because the error introduced by clouds is based on the synthetic dataset with a larger sampling of data than the original dataset of the error vectors. Designing the synthetic dataset to be statistically similar to the raw data is an advantage because it allows for more extensive testing than a fixed dataset allows. The likelihood of mismatches was nearly nonexistent for the clear features. This matches the expectations for a region in which the features were essentially unchanged with the cloud overlay. The likelihood in the edge regions provided the most interesting results, showing that the probability of a mismatch in edge regions smoothly and gradually increase as the feature similarity between two features decreases. The average likelihood of mismatching was small at only 5%, implying that it may be possible in the future for UAVs to navigate fairly accurately in an environment with semi-transparent clouds. The likelihood of mismatching in the cloud regions varied widely as a function of the distance between two features and had an average likelihood of 10%: twice that of the edges. Navigation in fully-obscured regions is therefore not very predictable, but as mentioned in Chapter I, it is assumed that the aircraft does not fly in such cloudy conditions. Therefore, features matched in a cloudy region could be ignored, as long as enough features in the clear and edge regions were identified.

5.2 Future Work

The method developed to determine the cloud-induced error in features was faced with several limitations throughout the process. For more accurate results that de-

scribe the true real-world scenario, these limitations must be addressed in future work on the research. To complete a more comprehensive study of the errors, a wider variety images must be tested. Given that the results were based off a single cloud template, some of the results may be specific to that particular cloud structure. Lastly, future endeavors should tie the results into a visual navigation system in order to be of practical use.

5.2.1 Feature Matching.

A major assumption made within the feature detection stage of the research is that the features on the test images were located in exactly the same locations as those on the truth images. The assumption was made so that features between the two image types could clearly be matched to each other for comparisons. In general, this assumption does not hold true except in the case of the clear features where there are no clouds to affect the descriptor calculations. However, without this assumption, features on the test images would likely be located in seemingly unrelated positions and matching features between images would be impossible without further investigation. Therefore, future work should examine the natural feature placement on the test images and determine a method in which to match features. Successful attempts at developing a feature matching method would be capable of producing results that more accurately describe the data.

5.2.2 Varying Landscapes & Flying Conditions.

As previously mentioned, the research uses a single cloud template and a single aerial image to generate results. In a real operation, different aircraft can be expected to fly over a variety of landscapes other than the same urban landscape used for this research. Using other landscapes as the aerial image, such as deserts or forests,

may produce different results. Future work could therefore determine if there is a relationship between the image scenery and the errors introduced by the clouds.

Since only a single cloud template was used for the research, the likelihood of errors was determined for a single altitude with a specific amount of cloud cover in the scene. Further research into the effects of clouds on visual navigation should take into account varying amounts of cloud cover and perhaps identify the maximum allowable percentage of cloud cover before the position solution becomes inaccurate to the point of being unusable. Additionally, adding the parameter of altitude into the research method would allow the method to adapt to the specific environment in which the aircraft is flying.

5.2.3 Interfacing with Visual Navigation System.

While the scope of this research limited the results to determining the likelihood of erroneous matches between features due to clouds, future work should expand upon the results and integrate them with a visual navigation system. As described in Chapter II, in order for the results to be incorporated into a visual navigation system, matches between sequential images in an image sequence must be matched and the image homography determined by using a method such as RANSAC. After registering the image using the homography, SLAM is used to update the position information of the vehicle. Because this research does not interact with a navigation system, the actual uncertainty in the aircraft's position is not determined. Incorporating the uncertainty measurements given by this research into the position information would allow an aircraft to determine the uncertainty in the navigation solution.

5.3 Summary

In summary, the research presented a method for determining the likelihood of mismatching features located in clear, edge, and cloud regions of an image. The method can be used for automatic testing of a visual navigation system to determine in which situations the system is likely fail and when it can be expected to succeed. Using this method as a testing tool will also allow further development in weak aspects of a visual navigation system that previously may have been undiscovered. Awareness of these situations will help to promote confidence in visual navigation techniques. Since the method is entirely simulation-based, money can be saved by simulating the desired environments instead of spending the resources to implement testing in a real-world environment.

Bibliography

1. D. G. Kottas and S. I. Roumeliotis, "Efficient and consistent vision-aided inertial navigation using line observations," in *IEEE International Conference on Robotics and Automation*, 2013, pp. 1540–1547.

2. G. Y. Garnder, "Visual simulation of clouds," *ACM SIGGRAPH Computer Graphics*, vol. 19, no. 3, pp. 297–303, 1985.

3. W. Dong, X. Zhang, and C. Zhang, "Generation of cloud image based on perlin noise," in *Multimedia Communications (Mediacom), 2010 International Conference on*, 2010, pp. 61–63.

4. M. Harris, W. Baxter, T. Scheuermann, and A. Lastra, "Simulation of cloud dynamics on graphics hardware," in *ACM SIGGRAPH/EUROGRAPHICS*, 2003, pp. 92–101.

5. N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *International Conference on Computer Vision & Pattern Recognition*, 2005, pp. 886–893.

6. P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001, pp. I511–I518.

7. D. Lowe, "Object recognition from local scale-invariant features," in *International Conference on Computer Vision*, 1999, pp. 1150–1157.

8. H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

9. L. Juan and O. Gwun, "A comparison of SIFT, PCA-SIFT and SURF," *International Journal of Image Processing*, vol. 3, no. 4, pp. 143–152, 2009.

10. K. Derpanis, "Integral image-based representations," York University, Tech. Rep., 2007.

11. C. Evans, "Notes on the OpenSURF library," 2009, online. [Online]. Available: http://www.chrisevansdev.com/computer-vision-opensurf.html

12. J. Barron, D. Fleet, and S. Beauchemin, "Performance of optical flow techniques," *International Journal of Computer Vision*, vol. 12, no. 1, pp. 43–77, 1994.

13. A. Sellent, D. Kondermann, S. Simon, S. Baker, G. Dedeoglu, O. Erdler, P. Parsonage, C. Unger, and W. Niehsen, "Optical flow estimation versus motion estimation," Universitat Heidelberg, Tech. Rep., 2012.

14. T. Wang, C. Wang, J. Liang, Y. Chen, and Y. Zhang, "Vision-aided inertial navigation for small unmanned aerial vehicles in gps-denied environments," *International Journal of Advanced Robotic Systems*, vol. 10, no. 276, pp. 1–12, May 2013.

15. X. Xu and S. Negahdaripour, "Vision-based motion sensing for underwater navigation and mosaicing of ocean floor images," in *OCEANS '97 MTS/IEEE Conference*, vol. 2, 1997, pp. 1412–1417.

16. M. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," SRI International, Tech. Rep., 1980.

17. H. Durrant-Whyte and T. Bailey, "Simultaneous localisation and mapping: Part I," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, June 2006.

18. K. Ohba and K. Ikeuchi, "Detectablity, uniqueness, and reliability of eigen windows for stable verification of partially occluded objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 9, pp. 1043–1048, September 1997.

19. J. Jurado, "Enhanced image-aided navigation algorithm with automatic calibration and affine distortion prediction," Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB OH, March 2012.

20. J. Morel and G. Yu, "ASIFT: A new framework for fully affine invariant image comparison," *SIAM Journal of Imaging Sciences*, vol. 2, no. 2, pp. 438–469, April 2009.

21. M. Villamizar, A. Sanfeliu, and F. Moreno-Noguer, "Fast online learning and detection of natural landmarks for autonomous aerial robots," in *IEEE International Conference on Robotics & Automation*, 2014, pp. 4996–5003.

22. G. Casella and R. Berger, *Statistical Inference*, 2nd ed. Stamford: Cengage Learning, 2002.

23. G. Terejanu, "Tutorial on Monte Carlo techniques," online. [Online]. Available: http://www.cse.sc.edu/~terejanu/files/tutorialMC.pdf

24. R. Kogler, E. Garutti, H. Stadie, and A. Schimdt, "Instrumentation and methods: The Monte Carlo method," online. [Online]. Available: http://www.iexp.uni-hamburg.de/groups/pd/sites/default/files/private/teaching/instrumentation-and-analysis-methods/MC_method1.pdf

25. GoldSim, "Monte-Carlo simulation: What is the Monte-Carlo method?" online. [Online]. Available: http://www.goldsim.com/Web/Introduction/Probabilistic/MonteCarlo/

26. A. Gutierrez and A. Jennings, "Cloud-induced uncertainty for visual navigation: Development of cloud templates," *Aerospace and Electronics Conference (NAECON), Proceedings of the 2014 IEEE National*, submitted for publication.

27. USGS EarthExplorer, "2005 ortho-rectified digital imagery for the Portland, OR metropolitan area," 2005, online. [Online]. Available: http://earthexplorer.usgs.gov/

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–12–2014	Master's Thesis	Oct 2013 — Dec 2014

4. TITLE AND SUBTITLE

Cloud-Induced Uncertainty for Visual Navigation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Gutierrez, Alyssa N, Ms.

5d. PROJECT NUMBER

14Y520C

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENY-MS-14-D-43

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory, Sensors Directorate
Attn: Dr. Clark N. Taylor
2241 Avionics Circle
WPAFB OH 45433
(937) 528-8184 (DSN 798-8184) Clark.Taylor.3@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RY

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Statement A. Approved for Public Release; Distribution Unlimited.

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the Unites States.

14. ABSTRACT This research addresses the numerical distortion of features due to the presence of clouds in an image. The research aims to quantify the probability of a mismatch between two features in a single image, which will describe the likelihood that a visual navigation system incorrectly tracks a feature throughout an image sequence, leading to position miscalculations. First, an algorithm is developed for calculating transparency of clouds in images at the pixel level. The algorithm determines transparency based on the distance between each pixel color and the average pixel color of the clouds. The algorithm is used to create a dataset of cloudy aerial images. Matching features are then detected between the original and cloudy images, which allows a direct comparison between features with and without clouds. The transparency values are used to segment the detected features into three categories, based on whether the features are located in the regions without clouds, along edges of clouds, or with clouds. The error between features on the cloudy and cloud-free images is determined, and used as a basis for generating a synthetic dataset with statistically similar properties. Lastly, Monte Carlo techniques are used to find the probability of mismatching.

15. SUBJECT TERMS

Visual Navigation, Uncertainty Quantification, Cloud Transparency Calculation, Cloud-Induced Error

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Alan L. Jennings, AFIT/ENY
U	U	U	U	118	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x7495; alan.jennings@afit.edu