Convolutional Neural Network-based Vision Systems for Unmanned Aerial Vehicles

Submitted by

Rytis Verbickas, B. Eng.

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of

the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical and Computer Engineering

Faculty of Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada K1S 5B6

December 2012

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-94243-7

Our file Notre référence
ISBN: 978-0-494-94243-7

Table of Contents

Abstract v

Acknowledgements vi

List of Tables vii

List of Figures viii

1 Introduction 12

1.1 Overview 12

1.2 Problem Statement 13

1.3 Thesis Contributions 13

1.4 Thesis Outline 14

2 Background 15

2.1 Low Flying Unmanned Aerial Vehicles for Geophysical Surveying 15

2.1.1 GeoSurv II 15

2.1.2 Telemaster Automatic Test Bed (ATB 10) 17

2.2 Autonomous Obstacle Detection and Avoidance 18

2.2.1 Obstacle Detection 19

2.2.2 Obstacle Detection Pipeline 22

2.3 Horizon Detection 23

2.3.1 Background 24

2.3.2 Support Vector Machines 30

2.4 Convolutional Neural Networks 31

2.4.1 Training 36

2.4.2 Evaluation 44

2.5 Stereoscopic Vision 45

2.5.1 Camera Model 46

2.5.2 Camera Calibration 47

2.5.3 Stereo Rectification 51

2.5.4 Stereo Correspondence 54

2.5.5 Disparity Computation and Depth Re-projection 61

2.6 Compute Unified Device Architecture 70

3 Convolutional Neural Networks for Horizon Detection 72
3.1 Experimental Setup 73
3.1.1 Convolutional Neural Network Training Framework 77
3.2 Network Output Representation 78
3.3 Dataset Expansion 80
3.4 Combining Sky and Horizon Masks 81
3.5 Evaluation 87
3.5.1 Performance Comparison with SVMs 100
3.6 Platform and Runtime Performance Considerations 102
3.7 Conclusions 104
4 Edge Feature Based Stereoscopic Obstacle Detection 104
4.1 Experimental Setup 105
4.2 Obstacle Datasets 105
4.3 Feature Extraction 109
4.4 Sparse Block Matching 112
4.5 Horizon Boundary Application and Model Construction 119
4.6 Runtime Performance Considerations 124
4.7 Conclusions 125
5 Conclusions 125
5.1 Summary of Results 125
5.2 Future Research 126
5.2.1 Horizon Detection 127
5.2.2 Obstacle Detection 127
5.2.3 Stereoscopic Vision 129
References 131

Abstract

Obstacle detection and avoidance for unmanned aerial vehicles (UAVs) is a challenging task, requiring processing speed and accuracy. Although a number of sensor solutions are available for this task, optical sensors are particularly suited being cheap, light-weight and long range. Stereoscopic systems with 2 cameras can be calibrated and used to perform localization of detected features in 3D space, allowing a model of the environment in front of the UAV to be constructed. Stereoscopic methods can, however, be computationally intensive and prone to mismatches which further increases the computational burden of a potential system.

This thesis proposes a new approach to horizon detection based on convolutional neural networks (CNNs) and uses knowledge of the sky, ground and horizon to simplify the search for potential obstacles to the horizon and sky region of an image. An edge feature based approach followed by stereo correspondence is then applied to detect and triangulate the location of potential obstacles, producing a 3D model as the system output which can be used by an obstacle avoidance algorithm to navigate the UAV.

Acknowledgements

I would like to thank my supervisor Professor Anthony Whitehead for a tremendous level of guidance through my research. This work would have been impossible without his direction and encouragement. I would also like to thank Professor Paul Staznicky for his work in managing the Carleton UAS project, of which I have been a part of, as well as Sander Geophysics Limited for carrying out test flights, sharing data as well as support for the project. Additionally, I wish to thank Shahab Owlia for his help and collaboration on the avionics and obstacle avoidance portions of the Carleton UAS project as well as Tom Hastie for constructing and flying the Telemaster test bed aircraft which allowed the data used in this thesis to be gathered.

List of Tables

Table 1 - Confusion matrix for summarizing the classification results on a sky-mask output generated by a network 44
Table 2 - Free parameter comparison between a convolutional and fully connected output layer 78
Table 3 - Effect of varying the number of feature maps at each layer 89
Table 4 - Rotation and translation effects on training performance 92
Table 5 - Training performance of a 246x246 to 116x116 4 layer CNNs on both datasets 94
Table 6 - Out of sample error for trained networks on 246x246 input images and 54x54 (4 hidden layer) and 116x116 (2 hidden layer) output images 96
Table 7 - Out of sample test performance for horizon-mask networks 98
Table 8 - Post processed sky-mask performance when combining sky and horizon mask networks 99
Table 9 - Performance Curve for Varying Input Sizes 103
Table 10 – Combined object detection rates for all video sequences (gradient threshold of 64) 110
Table 11 – Stereo detection rates (as percentage of total frames) for all manually identified obstacles (gradient threshold of 64, SAD window size of 5x5, disparity range 1-32) 119

List of Figures

Figure 1 - GeoSurv II UAV Prototype 16
Figure 2 – Sample images from a geophysical flight of an sUAS (courtesy of SGL) 17
Figure 3 - The Telemaster (ATB10) aircraft with mounted stereo cameras 18
Figure 4 – Obstacle Detection and Mapping Pipeline 23
Figure 5 - RGB image with corresponding sky-mask image 26
Figure 6 - Example of an RGB image and the corresponding 36 pixel high horizon-mask (18 pixels at and above and 18 pixels below the horizon boundary) 29
Figure 7 – A 5 layer + input CNN for horizon detection 33
Figure 8 – Example of neurons at a layer applying local receptive fields to neurons in an upstream layer 33
Figure 9 – Connection based representation of the previous figure, highlighting the receptive field connections for a single neuron 33
Figure 10 - Pinhole camera model [82] 46
Figure 11 - Single camera calibration process [82] 50
Figure 12 - Epipolar geometry of a stereo pair of pinhole cameras 52
Figure 13 - Original image pair (top) and stereo rectification result (bottom) 54
Figure 14 – Image from Telemaster flight at the Arnprior field (left) and the corresponding Sobel edge map (threshold 8, right). Balloon obstacles are circled in red for clarity. Image is inverted and highlighted in red for clarity. 58
Figure 15 – Left and right stereo image pair (top row) and corresponding color-mask images obtained from applying a Sobel edge mask to each (bottom row). Edge map is inverted and highlighted in red for clarity. 60
Figure 16 - Frontal parallel image pair pixel triangulation [82] 62
Figure 17 - Planes of constant disparity for a stereo camera pair [82] 64
Figure 18 - Plot of range resolution (Δz) versus the physical distance to a pixel (z) for different disparity resolutions 67
Figure 19 – Construction for estimating the physical dimensions subtended by a cameras field of view 68
Figure 20 – Center pixel (red) and its 8 connected neighbors 69
Figure 21 - CUDA thread hierarchy [93] 71
Figure 22 - Example images from dataset 1 (left) and dataset 2 (right) 74
Figure 23 – Example image from dataset 3 (SGL) showing distortions introduced by an interlaced camera 74
Figure 24 – Hyperbolic tangent activation function 77

Figure 25 - Example output for a trained network to an HSB color space input pattern (leftmost column). Second column: first convolutional layer, 5 feature maps. Third column: Fully connected output layer. Fourth column: Target output 80
Figure 26 - Example of the original image I and rotated image with inscribed square S (highlighted in white, thickened for clarity) 81
Figure 27 - Original RGB input image, the corresponding sky-mask image, a sky-mask where the proposed approach would produce an all sky result 82
Figure 28 - Corresponding sky and horizon masks. The horizon-mask is 40 pixels wide with 20 pixels above and 20 below the horizon 84
Figure 29 – Average MSE versus epoch for varying numbers of Hessian approximation patterns 91
Figure 30 – Sampling regions for performing "translation" data generation 92
Figure 31 – Example images containing low levels of light in the Arnprior dataset 97
Figure 32 – Stereo image pair from Arnprior video sequence (balloon obstacles location circled in red for clarity) 106
Figure 33 – Overhead image of the Arnprior airfield with approximate initial aircraft location (Y_1, red), end location (Y_2, red) and balloon obstacle location (O_B, blue). Distance scale shown bottom left. 106
Figure 34 – Aerial view of ground test location (distance scale, bottom left corner) 108
Figure 35 – Stereo image pair from a ground test sequence with above horizon obstacles circled and numbered 108
Figure 36 – Another stereo image pair from a ground test sequence with corresponding above horizon obstacles and occlusion circled and numbered 109
Figure 37 – Original stereo image pair (top) and the corresponding Sobel edge maps (bottom) with the balloon distinguishable above the horizon (gradient threshold of 64). Edge maps inverted for clarity 111
Figure 38 – Example image with could cover from the ORCC database (gradient threshold of 64). Edge map inverted and highlighted in red for clarity. 111
Figure 39 – Rectified stereo image pair from ground sequence #4 113
Figure 40 – Birchfield and Tomassi (BT) measure based disparity map (darker color is closer to the cameras as the image is inverted, disparity search range of 1-20 and edge gradient threshold of 64). 114
Figure 41 – ZNCC measure based disparity map (darker color is closer to the cameras as the image is inverted, disparity search range of 1-20 and edge gradient threshold of 64) 114
Figure 42 – SAD measure based disparity map (obstacles O_1, O_2 and O_3 from Figure 36 circled in red, darker color is closer to the cameras as the image is inverted, disparity search range of 1-20 and edge gradient threshold of 64) 115
Figure 43 – Rectified stereo image pair for the flight sequence 117
Figure 44 – Disparity map computed using SAD 5x5 block matching with annotated obstacles circled with respect to Figure 43 (disparity range 1-16, gradient threshold 64) 117

Figure 45 – Feature points used to estimate the validity of the re-projection..... 118
Figure 46 - Example collision scenarios with the collision obstacle forming part of the horizon boundary (left image: building and tree tops, right image: tree tops and poles) 120
Figure 47 - Example of an original image (left), a sky-mask (center) and horizon-mask (right) image generated by a CNN and up-sampled..... 121
Figure 48 - Example of an original edge map (left, gradient threshold 32), masked with the sky-mask in Figure 47 (center) and masked using a combination of sky and horizon masks (right). Images are inverted and highlighted in red for clarity. 122
Figure 49 – Example of an original image (left), a sky-mask (center) and horizon-mask (right) image generated by a CNN and up-sampled..... 122
Figure 50 – Example of an original edge map (left, gradient threshold 32), masked with the sky-mask in Figure 49 (center) and masked using a combination of sky and horizon masks (right). Images are inverted and highlighted in red for clarity. 123
Figure 51 – Disparity map with similar rectangles grouped by connected components outlined in red (9x9 SAD window, disparity range 1-20, edge gradient threshold of 32)..... 124

1 Introduction

1.1 Overview

The research in this thesis is aimed towards the GeoSurv II Unmanned Aerial Vehicle (UAV) project at Carleton University. This project is a collaboration with Sander Geophysics Limited (SGL) and aims to develop a fully autonomous terrain following UAV to carry out high resolution geomagnetic surveying. In order to be able to accomplish this task autonomously, onboard avionics must be able to process information on the surrounding environment and decide whether deviation from a pre-programmed flight plan is necessary; for example if an incoming obstacle is detected.

The task of accurately and repeatedly locating an object in a video sequence is an easy task for human observers but is generally difficult in the field of machine vision. It also represents only part of the problem, since detection of an obstacle does not provide knowledge of the obstacles 3D coordinates; detection itself is complicated by weather, cloud cover and variations in lighting. In the case of estimating object location, active sensor approaches are possible, such as laser or radar, as well as passive sensor approaches using a stereoscopic pair of cameras. Cameras are an attractive solution due to their compactness, relatively wide field of view in comparison to laser based solutions as well as their detection range compared to sonar based solutions.

In deploying a stereoscopic pair of cameras, features must be detected in order to triangulate depth information. Since the target platform is a UAV, it is clear in advance that a significant portion of the scene visible in the cameras will be sky. As the aircraft increases in altitude, this distinction will grow and in the process reduce the potential classes of obstacle which may be encountered. As such the region of interest for detection can be reduced to the horizon, in order

to encompass tall objects and the sky region above the horizon in order to focus on any moving objects. The first step in performing detection in these regions is to first extract them, using knowledge of their location to focus the computations of the obstacle detection and triangulation stages.

1.2 Problem Statement

This thesis focuses on obtaining an accurate estimate of the horizon and sky region of the image, performing obstacle detection and finally stereoscopic range estimation of detected obstacles. Regions of depth corresponding to obstacles are then further aggregated based on their proximity and range and used to construct bounding regions in 3D to represent each potential obstacle. Bounding regions are then passed to the obstacle avoidance processing stage, where they are further aggregated and used to compute the aircraft flight path.

1.3 Thesis Contributions

The contributions of this thesis are:

- The construction of a ground truth, flight captured, image dataset for evaluating the performance of potential horizon detection algorithms.
- Proposing a new approach to performing sky, ground and horizon boundary detection at a high rate of accuracy. The approach involves using convolutional neural networks (the first application of these networks to this task) to automatically learn spatial feature extractors for finding the sky and ground.
- Proposing and evaluating an above the horizon obstacle detection pipeline based on edge detection and stereoscopic vision which allows detected objects to be localized and mapped in 3D.

1.4 Thesis Outline

The chapters of this thesis are arranged as follows:

- Chapter 2: Provides an introduction to unmanned aerial vehicles (UAVs followed by a review of horizon detection, obstacle detection and stereoscopic vision in the context of UAVs. An obstacle detection processing pipeline is introduced which uses horizon detection as a starting point, in order to detect and map potential obstacles using a stereoscopic pair of cameras.
- Chapter 3: Proposes using Convolutional Neural Networks (CNNs) in order to perform accurate horizon detection as a first stage to obstacle detection.
- Chapter 4: Proposes an obstacle detection processing pipeline which uses stereoscopic vision in order to detect and map in 3D potential obstacles at or above the horizon line
- Chapter 5: Provides conclusions in regards to the concepts presented in the thesis and suggests recommendations for future research.

2 Background

2.1 Low Flying Unmanned Aerial Vehicles for Geophysical Surveying

UAVs constitute a wide area of research including geophysical surveying [1], environmental [2] and forest fire monitoring [3], border patrol [4] and numerous military applications. Their deployment is a result of their attractiveness in terms of low costs relative to manned aircraft [5], due to their automation and smaller scale compared to conventional aircraft, and especially, in the case of potentially dangerous applications, by reducing the exposure of pilots to dangers. The level of automation utilized by a UAV can vary by application but falls in a spectrum between an aircraft which is completely unmanned and flies with no supervision between takeoff and landing, or, an aircraft which may be remotely piloted by ground control personnel. In the case of a self-operating UAV, autonomy software must be present to administer a preset flight path, ensuring that any unexpected obstacles be detected and avoided while continuing operations to complete the preset mission.

2.1.1 GeoSurv II

The GeoSurv II, shown in Figure 1 below, is a UAV designed, constructed and maintained by the Carleton UAV undergraduate project in collaboration with Sander Geophysics Limited (SGL), an Ottawa-based geophysics company. The primary goal of the aircraft is geophysical survey missions; however the aircraft has been designed to be modular in order to allow for other potential mission scenarios. In particular, the GeoSurv is designed to be able to accommodate extra payloads in addition to its own onboard avionics systems [6]. These payloads are limited to 16 lbs. and 8.3 in^3, and can include magnetometers, cameras and processing hardware in order to augment or integrate with the existing avionics. The GeoSurv fuselage is 11.8 inches wide, 43

inches long and 15.3 inches high with a wingspan of 193 inches (4.9 meters). In addition, the aircraft is specified to be able to provide a minimum of 75W at 28V DC to both the onboard avionics and to any aircraft payloads. This should allow for more complex processing hardware, for example an x86 based system with an integrated GPU which could support both the obstacle detection and obstacle avoidance processing requirements.

Figure 1 - GeoSurv II UAV Prototype

The operational envelope of the GeoSurv for speed is 60 to 100 knots (30 – 51.4 m/s) while the operational altitude is between 10 and 8000 meters [7]; a mandated system requirement of the GeoSurv is that it have a laser altimeter to assess altitude above ground level (AGL). A previous flight test using a helicopter with a towed simulated unmanned aerial system (sUAV) was flown at heights of 50m, 100m, 150m and 200m where 50m represents a typical height which is expected to be flown, and which should provide enough resolution for the recorded magnetic field strength [8]. Typical images of encountered terrain during the 50m AGL flight are shown in Figure 2. At these heights potential obstacles could include silos, telecommunication and power towers, industrial chimneys and tall trees. The GeoSurv is not expected to fly in cluttered environments, such as urban areas with a number of buildings.

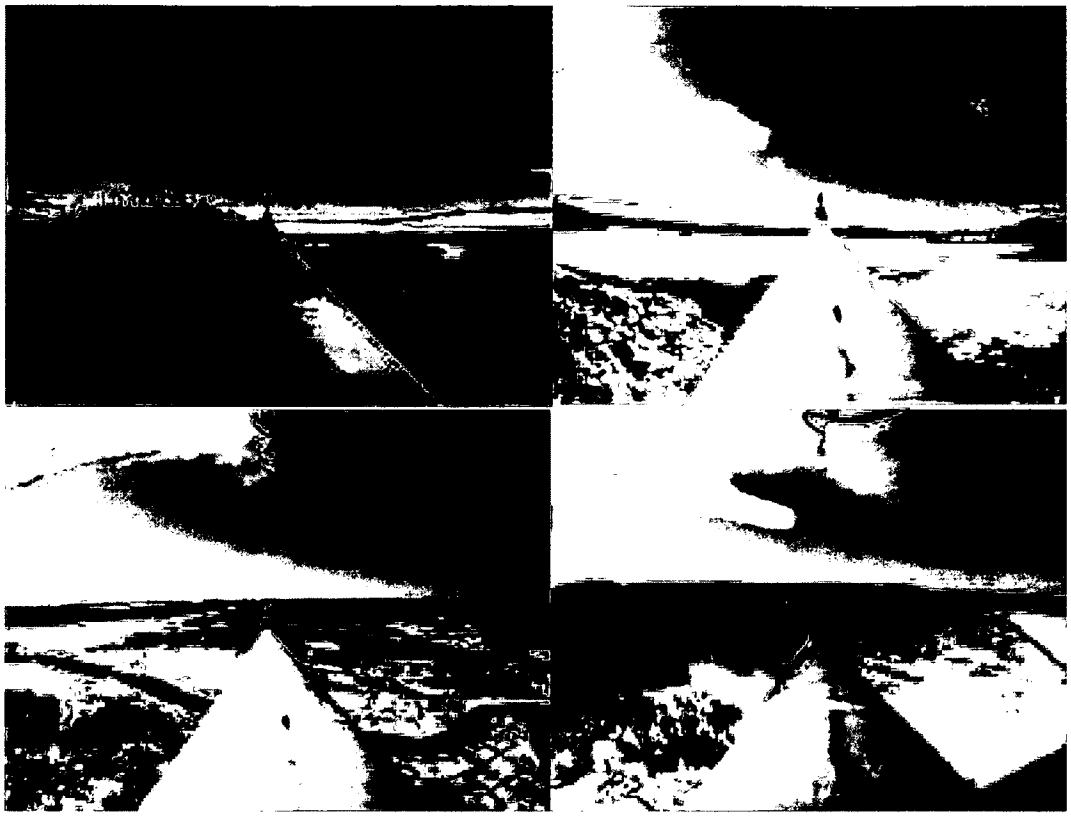

Figure 2 – Sample images from a geophysical flight of an sUAS (courtesy of SGL)

2.1.2 Telemaster Automatic Test Bed (ATB 10)

The Telemaster aircraft, shown in Figure 3 below, is an aircraft which serves as a test bed for new avionics and basic payloads such as cameras. The Telemaster contains an airspeed sensor and has a calibrated MicroPilot MP2028g autopilot which has recently flown autonomously with no RC operator intervention. The aircraft has been tested up to speeds of 30-40 knots and altitudes of up to 200ft. These limitations have mainly been imposed for safety, ensuring that the aircraft can be recovered in the event of an unusual attitude as well as making sure it stays in communication range. Most flight videos used in this thesis were gathered using cameras mounted to the wings of this aircraft, the specifications of which are detailed in Section 4.1.

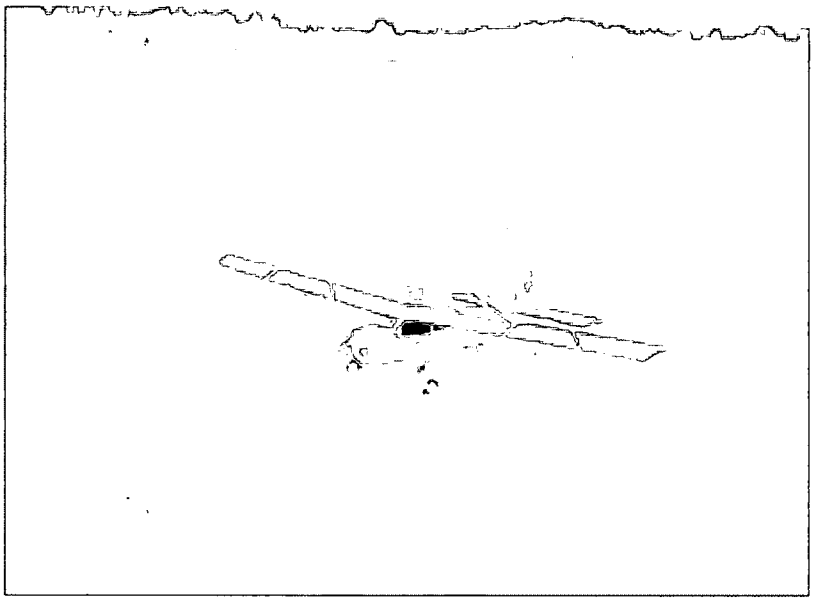

Figure 3 - The Telemaster (ATB10) aircraft with mounted stereo cameras

2.2 Autonomous Obstacle Detection and Avoidance

The GeoSurv II UAV has been created with the goal of performing autonomous geophysical surveying. To accomplish this task an autopilot with a number of sensors is to be deployed on the aircraft; one of these sensors is 2 forward facing cameras performing stereoscopic vision. In order to allow the aircraft to navigate successfully, the camera system must be able to perform detection and localization of obstacles, producing an output usable for the navigation and obstacle avoidance system guiding the aircraft. In this section we review common solutions to the problem of obstacle detection, including alternative sensors or camera configurations compared to our stereoscopic approach. We then summarize the type of output generated by our system and finish with a brief description of the obstacle avoidance portion of the processing pipeline and how the information from obstacle detection is utilized.

2.2.1 Obstacle Detection

In order for a UAV to perform autonomous obstacle detection and avoidance, some kind of active or passive sensing technology needs to be used. For this thesis the assumption is that the detection is performed using a stereoscopic pair of cameras mounted on the UAV. Stereoscopic based solutions to obstacle detection and avoidance are available in the literature, demonstrating that this approach is feasible for rotorcraft UAVs, miniature unmanned aerial vehicles (MAVs) as well as more general purpose, large scale, UAVs. In [9] Moore et al. describe the construction of a stereo vision based detection system which remaps the problem into a new cylindrical coordinate system to ease the computational burden of correspondence matching. Although the depth estimation performed is for every image pixel their system requires a specially constructed camera-mirror system in order to perform stereo correspondence. In [10], Achtelik et al. use FAST (Features from Accelerated Segment Test) features for stereo correspondence. FAST is a corner detection method similar to SIFT (Scale Invariant Feature Transform), Difference of Gaussians (DoG) or Harris corners, with the added advantage of having a faster run-time than the mentioned methods. The authors perform frame to frame feature tracking for each camera, using a pyramidal implementation of the Lucas-Kanade optical flow algorithm, as well as computing the stereo correspondence of features between both cameras. The end result is an MAV which is capable of sparse motion and depth estimation.

Solutions using off-the-shelf stereo systems, augmented by further post-processing, have also been investigated such as those in [11] and [12]. By construction these systems typically provide a dense depth map. In [11] Byrne et al. describe a segmentation based augmentation to the off-the-shelf Acadia I stereo vision processor. Their approach focuses strictly on below the horizon

detection and augments the stereo processor output with an additional global segmentation technique based on minimum graph cuts. The segmentation step refines the range estimates produced by the stereo vision processor in order to obtain a more accurate depth estimate. In [12], Hrabar used an off the shelf camera system with Stereo-On-A-Chip (STOC) technology for generating and updating a 3D occupancy grid of potential obstacles. Their investigation was limited to indoor, short range tests and focused mainly on evaluating their occupancy grid concept assuming an accurate disparity map as a starting point.

In the case where a stereoscopic pair of cameras is not feasible, such as for MAVs, monocular cameras have also been used both for obstacle detection [14] and also for computing depth from motion [14], [13], [16]. Depth from motion computes depth by tracking image feature points between successive image frames and inferring the 3D motion of the features based on their movement across an image. Since a-priori scene structure is typically not known, it is difficult to optimize the feature tracking processing. In addition, rapid changes to aircraft attitude, for example when preparing for or in the middle of an avoidance scenario, can introduce errors into feature tracking results [13].

When feature extraction is sparse, aggregation becomes necessary and can be performed using clustering techniques to obtain a denser estimate of objects or surfaces. Finally, in cases where feature tracking is not an option feature density distribution (FDD) is another approach to obstacle detection which estimates the time to collision for detected image features, such as edges, by looking at the rate of expansion in the image plane of the features [17]. Although the results presented by the authors in [17] evaluate the approach in a real scenario, their formulation assumes only a single obstacle in the field of view and straight aircraft trajectory with no changes in air-

craft attitude, the occurrence of which the authors acknowledge could severely degrade estimation results.

A combination of optical flow and stereoscopic vision, referred to as stereo-flow, has also been found to be useful in the navigation of urban canyons by a rotorcraft UAV [18]. There Hrabar et al. use sideways looking cameras for flow computations, generating turn commands to steer the UAV away from the side with the greatest recorded flow. On the other hand the stereo portion used an off-the-shelf solution which took care of camera calibration and rectification, providing a real-time depth map for the forward motion obstacle detection.

Alternatively, active range sensors are also available and include laser, radar or sonar based solutions. For UAV applications, laser based solutions typically break down into laser rangefinders and laser scanners. Laser rangefinders are focused, returning a single range estimate in a single direction [19][20]. In order to obtain a sweep however, special pan/tilt rigs need to be constructed [21]. Nonetheless, the range capabilities of these devices are impressive returning distance measurements of up to 400m. As alternative solutions to laser ragers, laser scanners can provide a wide field of view although it is typically still limited to a scanning plane. These solutions can even be lighter than traditional off-the-shelf STOC solutions, however for smaller laser sensors the maximum distance is quite limited; generally around 50m [22]. The advantage of laser systems over stereo camera systems is that accuracy typically does not vary much with range. Custom 3D laser radar scanner solutions with non-horizontal FOV's have also been tested on small rotor UAVs, being able to detect power lines at a distance of 150m [23]. Doppler radar systems have also been tested for feasibility in obstacle avoidance on a UAV [24], and although the mass of the radar unit (300g) and power consumption (<5W) was low, the range of the device was limited to 10-15m with an operational rate of 10Hz.

2.2.2 Obstacle Detection Pipeline

The overall processing pipeline for obstacle detection and 3D mapping is shown in Figure 4. The pipeline takes as input a stereoscopic pair of images and produces as output bounding boxes in 3D space representing detected obstacles. The bounding boxes contain groups of pixels found in close proximity containing similar depth. During this process the network performs segmentation of the image into 2 regions, one above the horizon and the other below, within the sky and horizon mask processing block; the details of this block are outlined in Section 3. As a result of this stage processing is focused on the region at and above the horizon; we rely on the unique properties of this region (compared to below the horizon) for this thesis. The subsequent stage of feature extraction extracts potential obstacle locations passing the processed images, with only regions of interest retained, to the rectification block which performs a mathematical manipulation of the images to prepare it for stereoscopic correspondence. The disparity map resulting from stereo correspondence is post-processed for noise removal, followed by a mapping step which projects feature points into 3D space as aggregated regions based on spatial proximity.

Since we expect to perform surveys during the daytime in favorable conditions, one of the assumptions is that we expect potential obstacles to have relatively high contrast to the surrounding sky region. This is a justified assumption since this would need to be satisfied to some extent in order to perform any feature extraction. The process can still be complicated with cloud cover, however we mitigate these effects by performing consistency checks in our range computations.

Although the assumption made in this thesis is that potential obstacles will be found at the horizon boundary or above it, it must be acknowledged that in certain scenarios (for example during landing or descent), obstacles may appear below the horizon and still pose a threat to the UAV. These scenarios are currently under consideration and will be addressed in the future as part of a

systems integration step. This step will combine several obstacle detection algorithms which, as a whole, will perform obstacle detection over the entire field of view.

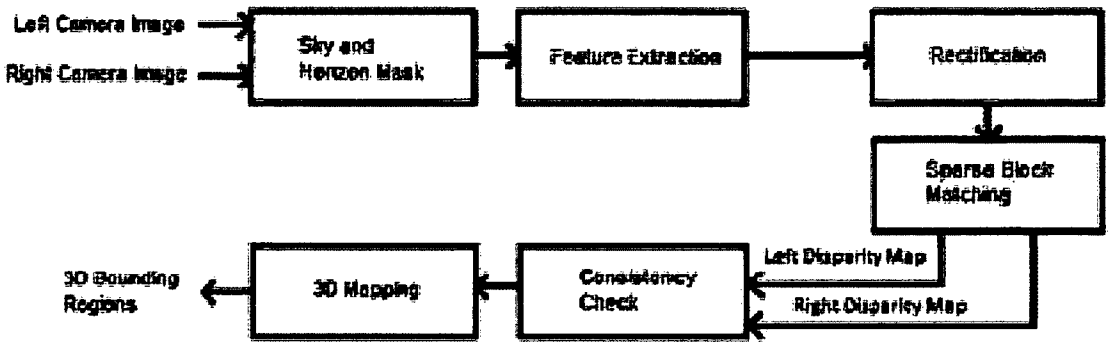

Figure 4 – Obstacle Detection and Mapping Pipeline

The obstacle avoidance portion of the processing pipeline is separate from the processing pipeline given in Figure 4, using the computed 3D bounding regions as input. All bounding regions are expanded or aggregated further based on requirements of minimum obstacle size in order to be able to compute an avoidance trajectory. Thus while the proposed obstacle detection pipeline does perform noise removal processing of depth information, it does not currently perform bounding region aggregation.

2.3 Horizon Detection

Horizon detection can be a first processing step for a UAV vision based system, for example in the case of attitude estimation [27] or obstacle detection [32]. In this thesis, horizon detection is used as a first step in the process of obstacle detection and as such the literature is reviewed in this section to detail the current approaches and corresponding limitations.

2.3.1 Background

A number of approaches have been developed in the literature for performing horizon detection using cameras mounted to varying forms of aircraft such as UAVs and MAVs. In the case of methods which focus on detecting the horizon boundary, we find a wide range of approaches which either make the assumption that the horizon boundary can be represented as a line as seen in [27],[29],[32],[36] and [37] or specify the exact division between the sky and ground as seen in [25] and [33]. For example in the case where the horizon is flat the former methods are sufficient, however when faced with highly varying terrain (at low altitudes) the latter methods are more appropriate. Generally, and in particular for this thesis as well, ground is referred to as anything which is below the sky region of the image including water, hills, trees and artificial objects. For methods which assume the horizon can be represented as a line, we predominantly see the Hough Transform [38] as a feature extraction technique, specifically in [25],[27],[29],[32] which is used in conjunction with a separate processing step such as edge detection [25],[29],[32],[36]. The straight line horizon boundary assumption is valid for a specific range of altitudes. When altitude is too high the curve of the earth will produce a curved horizon line. When the altitude is too low, especially in the case of a terrain following aircraft, obstacles and hills can also produce a horizon which is not a straight line.

Machine learning has also been applied as a preliminary step in horizon boundary detection. In [32] McGee et al. used a support vector machine (SVM) along with edge detection and the Hough Transform on a video sequence taken from onboard a low flying aircraft; the Hough Transform is applied to detected edges in order to find the longest, or most dominant, edge. The feature inputs classified by the network were raw pixel values in the YUV color space after

smoothing with a Gaussian filter. The classified image was further processed using dilation and erosion operators followed by edge detection. The edge map then underwent a voting process and minimization of a cost function to find the single dominant line appearing in the image which was selected as the horizon line. Although the authors were not checking the pixel-wise classification accuracy of the SVM, as they were interested in the horizon line, they report that the correct horizon line was found in 90% of frames with difficulty observed in cases where illumination changes occur such as when the aircraft changes pitch angle. In [34], Boroujeni et al. used segmentation to detect the presence of a light field in order to detect the horizon. The authors investigated 2 clustering approaches to do this, namely intensity based and k-means clustering of the color information in the image. Detected clusters were checked in order to determine whether they span from one end of the image to the other, with the thinnest cluster selected as the winner. Their approach rests on the assumption that a valid light field is present for all images during flight for example with cloud cover, camera exposure adjustment and illumination changes. Further to this the horizon is assumed to span from a particular side of the image to another, although depending on the current maneuver being performed this may not be the case. This is especially apparent in cases where only sky or ground is visible, which could produce very erroneous results.

In contrast to the approach which focuses on finding only the horizon boundary, methods which classify all pixels as belonging to either sky or ground can also be found [26],[28],[30],[31],[32], [34],[35]. These approaches don't make any simplifying assumptions on flight altitude or horizon shape. The result is a kind of "sky-mask" which allows one to selectively pick the sky or ground regions from the original input image. An example is shown in Figure 5 below with the corresponding input image.

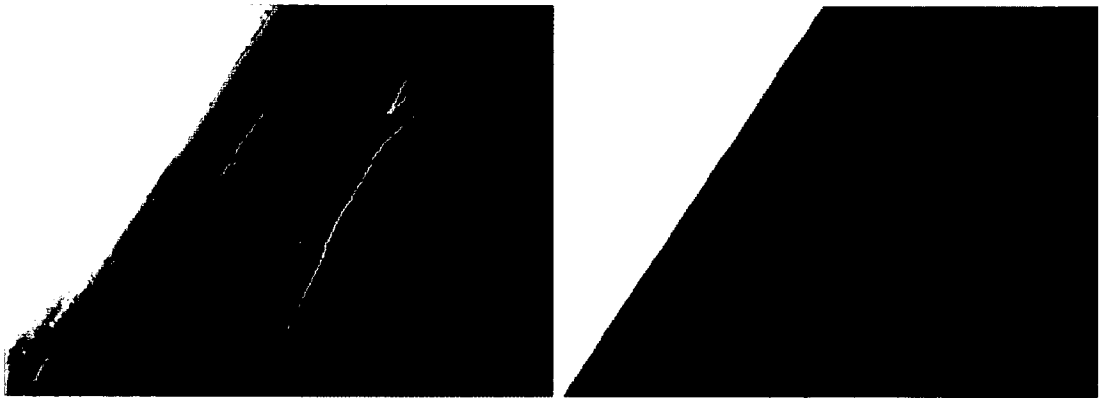

Figure 5 - RGB image with corresponding sky-mask image

Generating a sky-mask involves classifying image pixels and typically relies on machine learning techniques to generalize the difference between sky and ground pixels [28],[30],[32],[34],[35], allowing a classification to be made, however hand-crafted techniques can also be found [26][31]. Classification of pixels typically breaks down to either using pixel intensities for features or a number of statistics computed around a neighborhood of each pixel or a combination of both. In [31], Minwalla et al. classify pixels into sky and ground based on the weighted contribution of a number of parameters in a small 11x11 area around each image pixel. These parameters included min, max, mean, smoothness and a number of Gabor filters [39] at select orientations. Each parameter was then fit with 2 Gaussians, one for sky and one for ground, with thresholds selected to separate both distributions across all parameters. Although the authors mention that their Gaussian assumption held for tested images, this may not be the case in general with scene illumination changes or varying aircraft maneuvers. The weight contribution of each filter was hand tuned, which could present a problem in large and diverse datasets.

In the case where machine learning methods are applied for learning the classification of sky and ground we see a number of potential solutions. In [28] Shinzato et al. use artificial neural net-

works (ANNs) to perform sky and road detection. They divided an input image into a grid of smaller images, constructing a feature vector for each sub-image using metrics such as mean, entropy, variance and energy. These metrics were computed in varying subsets of channels from the RGB, HSV and YUV color spaces. An ensemble of 4 multi-layered perceptron networks was then trained using the known label of sky and road for all sub-images in the image training set. This approach has the drawback that when sub-images contain the horizon line the expected output of a network is unclear. It does however avoid a common problem with simple multi-layered perceptron networks which is their lack of invariance to input changes. In [34] de Croon et al. considered sky segmentation for obstacle detection, using a decision tree with a number of input features; these were a number of statistics such as mean, variance and entropy computed over image patches centered at all pixels. As the authors were trying to evaluate the best features, they investigated subsets of all possible features. They favored smaller decision trees in an attempt to learn better generalizations. Decision trees have the added benefit of being easy to read once trained, allowing one to see exactly what the network is basing its decisions on. In addition not all features are necessarily used to perform the final classification and only features on the evaluation path through the tree need to be computed for an input image, saving computational cost.

The work in [34], particularly the features which were used, were based in part on the work of Fefilatyev et al. in [30] who compared the relative performance of support vector machines (SVMs), decision trees and Naïve Bayes classifiers for sky-mask detection. Unfortunately this evaluation was small with only 2 groups of 10 images with the ground portion of all images consisting of water, making it difficult to infer generalizations of the performance of the tested classifiers. The first group was similar images, taken with the same camera at the same time of day but with varying amounts of sky visible while the second group was a more diverse and harder

set of images in varying lighting conditions. Performance on the easy dataset was noticeably higher, 98.19% classification accuracy for the best classifier (SVM), than for the hard dataset where the best average performance fell to 92.7% accuracy (again the SVM classifier). Another approach, proposed by Todorovic et al. in [35], uses the Complex Wavelet Transform (CWT) as a feature extractor with a Hidden Markov Tree (HMT) to represent the underlying statistical models over the feature space. Their approach relies on color and texture in the scene, specifically hue and intensity for color and the CWT for texture. Unfortunately their evaluation is quite short producing example outputs of only 3 images.

The above mentioned methods can be broken down into 2 different classifications. For this thesis we refer to the 2 approaches as horizon-mask and sky-mask based methods. Horizon-mask methods are seen to focus specifically on the dividing region between the sky and the ground. If, image column by image column, we place a single mark at the lowest sky pixel seen in the corresponding column the result will give us the horizon boundary. Certain columns may not contain any sky pixels which we leave blank or may contain only sky pixels which we also leave blank. The traced boundary can vary drastically in shape, may be missing in the case of extreme aircraft attitudes, or may be the special case of a straight line (a horizon line). To justify the name horizon-mask however, we consider the more general case of placing one or more marker pixels per column in the region of the horizon, an example of which is shown in Figure 6 below. The transition between sky and ground, or horizon boundary, is inside the mask area and in the case where the horizon-mask is 1 pixel thick is just the horizon boundary. The alternative, or sky-mask, methods instead mark all sky pixels with a particular color and ground pixels of a different color. An example of an input image and its corresponding sky-mask was shown earlier in Figure 5. Knowledge of the individual regions allows one to selectively mask the desired regions of each

image; for example in the case where one wants to focus on the sky portion of the image for obstacle detection. The distinction of a horizon-mask is necessary in this thesis for the purposes of training a neural network, which requires sufficient width in the target output horizon boundary in order to be trainable to generate the same output.

Figure 6 - Example of an RGB image and the corresponding 36 pixel high horizon-mask (18 pixels at and above and 18 pixels below the horizon boundary)

As mentioned, for horizon-mask based approaches, we find several rely on edge detection to narrow the horizon location in the image and require further post processing. This is a direct result of the general nature of the corresponding edge detectors which simply find vertical or horizontal edges. If instead one were to sacrifice the generality of the basic edge detectors and devise specialized spatial filters suited for detecting only the horizon line, the amount of post-processing required should decrease. However it is desired for automatic learning of specialized filters based on example images since handcrafting them would be a challenge; the expectation may be that such filters should still end up resembling simple edge detectors. As mentioned traditional learning methods for this task use either image pixels or metrics computed over image windows, however these methods simply perform classification. In order to learn general, shift and rotation invariant filters, we instead apply Convolutional Neural Networks (CNNs) for this task since

they are particularly suited for supervised learning of spatial filters by construction. We apply the same reasoning to the task of learning to produce sky-mask images, by training CNNs to learn to classify whether particular pixels are sky or ground.

2.3.2 Support Vector Machines

For this thesis, Support Vector Machines (SVMs) are used as the main competing learning model to CNNs for sky-mask generation. Existing work using SVMs for sky-mask generation is very sparse, with the work in [30] being the only example; the work there was restricted to only a handful of training images but showed that SVMs, on average, outperform decision trees or naïve Bayes classifiers for the task. In addition, the large majority of literature focusing on sky-mask generation use machine learning methods and because SVMs are a state-of-the-art classification tool with a wide variety of additional, and very successful, applications including handwritten digit recognition [40][42], object recognition [43] and face detection [44]. In addition, highly optimized toolsets in a number of different languages are available to train SVMs [45][46]; the evaluation for this thesis was carried out using the MATLAB 2012b SVM implementation for simplicity.

Support Vector Machines are a supervised learning model particularly suited for binary classification. This suitability comes from their mathematical formulation, which casts the problem of separating a training dataset, which consists of positive and negative examples, into an optimization problem. The optimization problem focuses on constructing the maximum separating hyperplane between the positive and negative examples in the training dataset, which can be constructed through a linear combination of the training examples closest to the separating hyperplane; these training points are called support vectors.

The optimization process of finding the maximum margin hyper-plane can be carried out using quadratic programming (QP) [47] or Sequential Minimal Optimization (SMO) [48]. Extensions to the basic linear SVM include the soft-margin SVM which allows for some amount of violation of the support boundary [40] as well as the kernel-trick [41] which maps the original feature space into a higher, and possibly infinite, dimensional space and performs the maximum margin separation there. The kernel-trick can be applied using any viable kernel function, including polynomial and radial basis kernels. An important consequence of SVMs is that the expected value of the "out of sample error" (data points outside of the training set) is bounded by the expected value of the number of support vectors (determined by performing leave one out training) and the size of the training dataset.

In order to classify pixels in an image as either sky or non-sky using an SVM, a number of features need to be extracted for each original pixel. The features used for experiments were those described in [30], which included the intensities of each pixel in the input image (for all color channels), as well as the mean, standard deviation, smoothness, third moment, uniformity and entropy of a small window around each pixel also computed for each channel. These measures ensure that 21 features are used for 3 channel images and 7 features for grayscale.

2.4 Convolutional Neural Networks

Convolutional Neural Networks are a biologically inspired variant of multi-layered perceptron networks (MLP's), specialized for image processing [49]. First popularized by LeCun et al. in [49] they're similar to other hierarchical feature extraction methods such as the Neocognitron [50] and HMAX [51][52]. These approaches vary in their network construction and initialization; however they're all reminiscent of the early work of Hubel and Wiesel who pioneered the

idea of simple and complex cells through their work with the visual cortex of a cat [53]. The basic idea of such architectures is to employ alternating layers of simple and complex cells, where simple cells are simple feature extractors and are followed by complex cells which locally pool the results of simple cells. In performing the pooling operation, such architectures gain some invariance to small distortions in the input pattern such as rotations and translations. Moving deeper into such a network, we find higher level representations of the original input which then allow us to make some kind of classification of the input pattern.

CNNs also follow a similar architectural pattern; however feature extractor learning is done automatically either through supervised or unsupervised learning methods. They have been applied to achieve state-of-the-art performance in a multitude of tasks including handwriting recognition [54] and face detection [59]. The structure of a typical CNN is shown in Figure 7. They consist of alternating layers of convolutional and pooling followed by an output classification layer. Each type of layer contains several feature maps, or groups of neurons, in a rectangular configuration. The neurons in a particular feature map each apply small receptive fields to regions in feature maps in the previous layer based on their position in their feature map as shown in Figure 8. The receptive field itself is simply a number of weighted connections as shown in Figure 9, that is, each connecting edge has a weight. The group of weights applied by a neuron is called a weight kernel. A distinguishing property of these networks is that all neurons in a feature map share the same weight kernel. The idea behind this configuration is that a spatial feature detector should be useful across an entire image, instead of just at a particular location; for example a vertical edge detector. The convolutional layers in the network perform the majority of processing in these networks, with the feature maps in the pooling layers simply down-sampling their corresponding feature map in the convolutional layer.

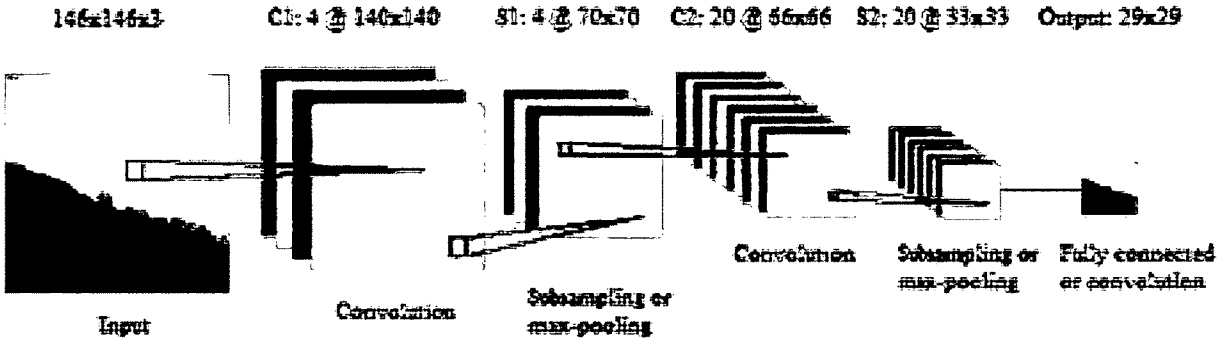

Figure 7 – A 5 layer + input CNN for horizon detection

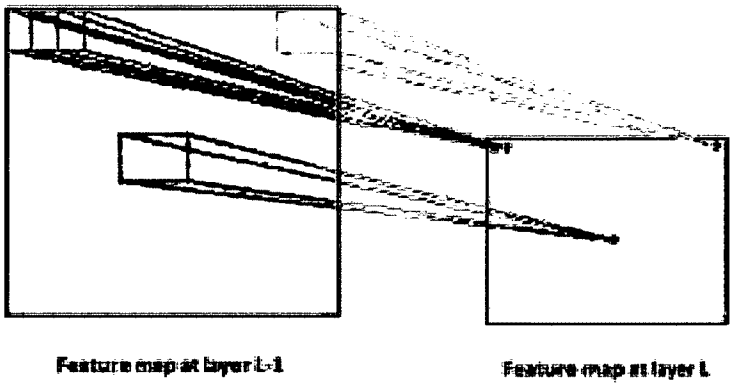

Figure 8 – Example of neurons at a layer applying local receptive fields to neurons in an up-stream layer

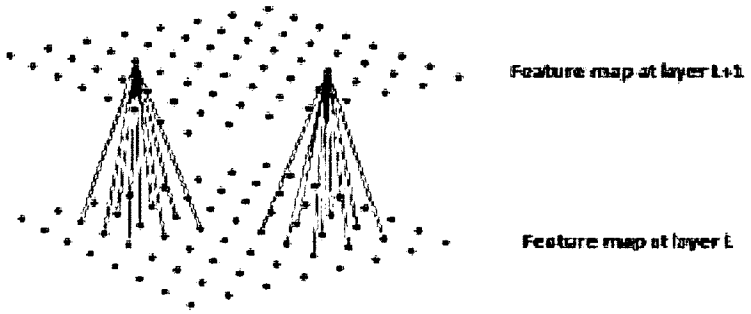

Figure 9 – Connection based representation of the previous figure, highlighting the receptive field connections for a single neuron

$$a_{m,n}^{L,k} = \tanh\left(\sum_{B_p \epsilon B_{L,k}} \sum_{(m,n,i,j)\epsilon B_p} a_{m,n}^{L-1,q} w_{i,j}^{q} + b_k \right) \qquad \text{Equation 1}$$

The convolution operation in a CNN can be written as shown in Equation 1. There we have $a_{m,n}^{L,k}$ as the activation of the neuron at spatial coordinate (m,n) in feature map k in layer L. If we take $B_{L,k}$ to be the set of receptive fields of the k-th feature map in the L-th layer and let B_p be the p-th receptive field (that is, at the p-th feature map) at layer L-1 where the elements of B_p are 4-tuples which specify the coordinate of each connection at the previous layer (m,n) and the corresponding coordinate of the connection within the receptive field (i,j). We then compute a weighted sum of the activations (from the previous layer) in the receptive field, add a trainable bias and pass the result through a non-linearity. Note that for adjacent neurons at layer L and feature map k, the corresponding receptive fields are also adjacent, simply shifted over by one unit.

Two popular pooling approaches are max-pooling and sub-sampling. The max-pooling operation is implemented in its own layer, with every feature map at the max-pooling layer assigned to a single feature map in the previous layer. The max-pooling operation simply selects the maximum activation in the receptive field of a particular neuron such that the neuron in the max-pooling layer produces the corresponding activation. The operation can be written as:

$$a_{p,q}^{L,k} = \max_{(m,n,i,j)\epsilon B} (a_{m,n}^{L-1,k} u(i,j)) \qquad \text{Equation 2}$$

Here the notation is the same as given previously for convolution, however u(i,j) is the weighing window function that multiplies the activation of the corresponding neuron in the receptive field. Pooling windows can be of any size and can be either overlapping or not. Furthermore the window function u(i,j) does not have to be rectangular and can be more complicated such as a Gaussian. Experimentally, the max-pooling operation has been shown to outperform the sub-

sampling operation on real-world datasets [61][62], however pooling operations other than the simple rectangular function as well as overlapping receptive fields have been shown to decrease overall performance.

The subsampling operation is also a pooling operation which averages the activations of the neurons in its receptive field using the relationship in Equation 3 below. The definition of activation variables, receptive field B and indeces (m,n) are the same as for max-pooling however now a trainable weight w_k and trainable bias b_k are shared by all neurons within the particular feature map. The weight multiplies the accumulated activation adds a bias and passes it through some nonlinearity. The receptive fields in a subsample layer are non-overlapping, which has the effect of down-sampling an image by the dimension of the receptive field.

$$a_{m,n}^{L,k} = \tanh(w_k \sum_{(m,n)\in B} a_{m,n}^{L-1,k} + b_k) \qquad \text{Equation 3}$$

Although sub-sampling layers have a one to one correspondence with feature maps in a previous layer it is also possible to integrate the convolution and subsampling operation together. When a convolution is performed by a neuron, if the adjacent neuron applies a receptive field which is shifted by more than 1 unit, for example 2 units, then the resulting feature map is effectively half the size of that in the previous layer. This architecture has been investigated in the literature and has shown comparable performance to traditional convolutional and subsampling CNNs [54] for handwriting recognition.

2.4.1 Training

Due to their similarity to Multi-Layered Perceptron (MLP) networks, CNNs can be trained with gradient descent using a small modification to the standard back-propagation (BP) algorithm [49][60]. In order to evaluate the network error for an input pattern and be able to perform BP we use the mean squared error (MSE) measure. Although other metrics are possible [54] we use MSE for its simplicity. For a vector of network weights w, let J(w) be the network MSE.

$$J(w) = \frac{1}{2}\sum_{k=1}^{e}(t_k - z_k)^2 \qquad \text{Equation 4}$$

Here we assume $w \in R^u$ where u is the total number of network weights and biases. For training pattern (x_m, t_m) where $x_m \in R^q$ is the m-th input pattern and $t_m \in R^e$ is the corresponding target output pattern which we expect the network to produce due to x_m and q and e are the number of input features and network outputs respectively. For a given x_m and weight vector w the network will produce an output $z_m \in R^e$.

It is important to note that we're computing J(w) for a single input pattern which is known as stochastic back-propagation. This is because a training pattern is selected at random without replacement from the training set and is back-propagated with the network weights also updated; one epoch is the random selection and back-propagation of all patterns. It is also possible to compute the average, or batch, gradient in the network by randomly selecting a small group of training patterns and computing an average gradient. This has the effect of producing a less-noisy measure of the gradient than stochastic back-propagation, however the cost is more computation. In choosing batch optimization of the network, we could select more efficient methods than simple gradient descent such as conjugate gradient or quasi-Newton methods. However, in

the case where there are redundancies or duplication in a dataset, stochastic gradient descent has been shown to be efficient in training neural networks [49][55].

For gradient descent we need to compute the gradient of the network error with respect to the network weights. This gives us the direction of maximum increase in the network weights which can be written as:

$$\Delta w = -\eta \frac{\partial J}{\partial w} \qquad \text{Equation 5}$$

and used to update the current weight vector w to form a new vector w' as:

$$w' = w + \Delta w \qquad \text{Equation 6}$$

We take the negative of the gradient since we wish to update our weights in the direction of maximum decrease and multiply by a small learning rate η which helps to scale the size of the change in the network weights. This is an important parameter in controlling the performance of gradient descent, the selection of which will be discussed shortly. In order to compute the gradient efficiently we can use the standard BP algorithm with a small modification for CNNs.

$$\frac{\partial J}{\partial w_{k,j}} = \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial w_{k,j}} \qquad \text{Equation 7}$$

Here the gradient descent equation has been expanded using the chain rule for neuron k at layer L and neuron j at layer L-1, with weight $w_{k,j}$ between them, the output of neuron k as z_k and net activation as net_k. We can further expand these equations using the following definitions, where $f(net_k)$ is the activation function for all network neurons:

$$z_k = f(net_k) \qquad \text{Equation 8} \qquad net_k = \sum_{j=1}^{H} w_{k,j} z_j \qquad \text{Equation 9}$$

$$\frac{\partial z_k}{\partial net_k} = f'(net_k)$$ Equation 10

$$\frac{\partial net_k}{\partial w_{k,j}} = z_j$$ Equation 11

These equations simplify Equation 7 to:

$$\frac{\partial J}{\partial w_{k,j}} = \frac{\partial J}{\partial z_k} f'(net_k) z_j$$ Equation 12

At the output layer, taking the derivative of the MSE equation gives:

$$\frac{\partial J}{\partial z_k} = -(t_k - z_k)$$ Equation 13

This with the simplified form of Equation 12 gives a simple expression for computing the weight deltas of the fan-in connection to the neurons at the output layer. We can further write an expression for the sensitivity of each output neuron k to be:

$$\delta_k = -\frac{\partial J}{\partial net_k} = (t_k - z_k) f'(net_k)$$ Equation 14

The sensitivities of each neuron at a particular layer in the network is the information which is actually "back propagated" through the network to the upstream neurons. Knowing the sensitivity of a neuron at layer L and the output of a particular neuron at layer L-1 (assuming a sigmoid or hyperbolic tangent activation function) allows one to compute the weights update for the corresponding connection. Finally, for neuron j at layer L-1 (upstream from the output layer) and neuron i at layer L-2, we have:

$$\frac{\partial J}{\partial w_{j,i}} = \frac{\partial J}{\partial z_j} \frac{\partial z_j}{\partial net_j} \frac{\partial net_j}{\partial w_{j,i}}$$ Equation 15

where the $\frac{\partial z_j}{\partial net_j} \frac{\partial net_j}{\partial w_{j,i}}$ term is computed as before but $\frac{\partial J}{\partial z_j}$ becomes:

$$\frac{\partial J}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\frac{1}{2}\sum_{k=1}^{e}(t_k - z_k)^2\right) = -\sum_{k=1}^{e}(t_k - z_k)\frac{\partial z_k}{\partial z_j}$$

$$= -\sum_{k=1}^{e}(t_k - z_k)f'(net_k)w_{k,j} = \sum_{k=1}^{e}\delta_k w_{k,j} \qquad \text{Equation 16}$$

Where the sensitivity of an upstream neuron, δ_j, can be written as:

$$\delta_j = \frac{\partial J}{\partial z_j}\frac{\partial z_j}{\partial net_j} = \left(\sum_{k=1}^{e}\delta_k w_{k,j}\right)f'(net_j) \qquad \text{Equation 17}$$

And so these equations allow us to compute weight updates for the output layer and any interior network layer simply with information from a forward pass through the network for a pattern, a target output and the sensitivities of the network which are computed in the upstream direction. The update to the network bias is computed in a similar manner and is simply the sensitivity of a particular neuron.

In the case of CNNs the modification to this algorithm is small and simply relies on the observation that all neurons in a feature map share the same set of weights, which means to compute the update for a single weight we must sum the weight delta contribution over all neurons in a feature map; it is the same for the bias.

$$\frac{\partial J}{\partial w_k} = \sum_{(i,j)\in S_k}\frac{\partial J}{\partial u_{i,j}} \qquad \text{Equation 18}$$

If we take a neuron at a feature map in layer L, then we have some number of weights which we can index as w_k and we have the network gradient as shown in Equation 18. There S_k is the set of indexed pairs of neurons, denoted by index i for the upstream neuron and index j for the downstream neuron, which share weight w_k and $u_{i,j}$ represents the connection weight; indeed we have $w_k = u_{i,j}\ \forall(i,j) \in S_k$. Thus the contribution to the gradient for a weight is summed

over all connections which use the weight. As mentioned the bias for a neuron is simply the sensitivity of the neuron. Since the bias is shared across the neurons in a feature map, similar to the weight modification for CNNs, the sensitivities of all neurons can be summed to compute the bias delta. Finally the sensitivity of a neuron is back-propagated to all neurons in the upstream layer which are in the downstream neurons receptive field. The back-propagation is done using the corresponding weights for each connection.

Using the process summarized above we can train a CNN to learn feature extraction filters in a supervised fashion. The outstanding question is how to compute the learning rate η. In a naïve implementation, we choose a starting value for η and reduce it incrementally as we train. This can cause problems in cases such as descending into a basin where we may "jump out" if the learning rate is too high or along a smoothly sloped surface where we'd like to take large steps to make sure convergence does not take too long. A constant learning rate also assumes that the contribution of each weight to the network gradient is equal. This is not necessarily justified since the error function may deform differently in different directions in the weight space. In order to improve performance we can use information in addition to the gradient, such as knowledge of the curvature of the error surface known as second-order information, to speed up convergence.

A simple way to estimate second-order information with minimal computation is to use momentum [56]. More computationally intensive measures to compute the optimal learning rates also exist. Arguably the most popular of these is the Levenberg-Marquardt algorithm and its many variants [56]. The main idea behind the Levenberg-Marquardt method is to perform a quadratic approximation of the error surface of a function in a local neighborhood of the error surface [57]. In doing so the weight update for a network becomes

$$w' = w - H^{-1}\nabla J \qquad \text{Equation 19}$$

and can be further written, as in the Levenberg algorithm, as:

$$w' = w - (H + \mu I)^{-1}\nabla J \qquad \text{Equation 20}$$

or in the more complicated form, as in the Levenberg-Marquardt algorithm, as:

$$w' = w - (H + \mu diag(H))^{-1}\nabla J \qquad \text{Equation 21}$$

where the matrix H is the Hessian matrix of second derivatives of the network error with respect to the weights, I is the identity matrix and diag(H) is a diagonal matrix consisting of the diagonals of the Hessian matrix. By adjusting the size of the trust region (μ) a tradeoff can be made between regular back-propagation (large μ) and the Hessian or Gauss-Newton approximation (small μ). As we can see from Equation 19, computing the optimal learning rates in the network requires the computation of the Hessian matrix. Although this can be done in practice, it can be beneficial to approximate the Hessian matrix, H, using the product of Jacobian matrices, J, of first derivatives of the network error with respect to the network weights; this gives the so called Gauss-Newton approximation which can be expressed as $H \sim J^T J$.

Algorithms exist to efficiently compute the second-order derivatives in a neural network using back-propagation [55]. Ideally, the Hessian and Jacobian matrices are computed over all training points in the dataset whenever it is required during training. However [49] makes an approximation where only a random subset of the total training set is used for computation, and only at the onset of a particular training epoch. This is a reasonable assumption if the second-order properties of the error surface change slowly and simply needs to be confirmed by experimentation. In our application, since we are facing potentially large networks with large image datasets this ap-

proximation seems justified. The network gradient J in Equation 4 can then be modified to give the following equation for the second derivative of the gradient with respect to weight $w_{i,j}$:

$$\frac{\partial^2 J}{\partial u_{i,j}^2} = \frac{1}{P}\sum_{p=1}^{P}\frac{\partial^2 J^p}{\partial u_{i,j}^2} \qquad \text{Equation 22}$$

where we now have an average MSE over a number of randomly selected patterns P with J^P the error for a particular pattern p and $u_{i,j}$ is taken as before to be the weight for a particular connection from neuron i at layer L to neuron j at layer L+1. Expanding the summation term in Equation 22 using Equation 18 gives Equation 23, where the second derivative with respect to the net activation can be simplified to give Equation 24:

$$\frac{\partial^2 J^p}{\partial u_{i,j}^2} = \frac{\partial^2 J^p}{\partial net_i^2} z_j^2 \qquad \text{Equation 23}$$

$$\frac{\partial^2 J^p}{\partial net_i^2} = f'(net_i)^2 \sum_v \sum_b w_{v,i} w_{b,i} \frac{\partial^2 J}{\partial net_v \partial net_b} + f''(net_i) \sum_v w_{v,i} \frac{\partial J}{\partial net_v} \qquad \text{Equation 24}$$

If we neglect the off diagonal elements of the Hessian, Equation 24 simplifies to Equation 25 as follows

$$\frac{\partial^2 J^p}{\partial net_i^2} = f'(net_i)^2 \sum_v {w_{v,i}}^2 \frac{\partial^2 J}{\partial {net_v}^2} + f''(net_i) \sum_v w_{v,i} \frac{\partial J}{\partial net_v} \qquad \text{Equation 25}$$

This simplifies the computation of the network learning rates significantly; however the second term in Equation 25 can still be negative. This means at times we may be exploring "uphill" in-

stead of "downhill". The second term can also be dropped in order to get a valid, strictly positive, approximation which happens to be the Gauss-Newton approximation to the Hessian matrix, albeit with the diagonal terms neglected. This approach is attractive when considering computational complexity, with the computation now very similar to the term for first order back-propagation in Equation 16. Having only diagonal Hessian terms also greatly simplifies the computation of the inverse Hessian since the inverse of a diagonal matrix is simply another diagonal matrix with inverted diagonal elements. We can then express the learning rate, given in Equation 5, for each weight in the network as follows:

$$\eta_k = \frac{\lambda}{\mu + h_{kk}} \qquad \text{Equation 26}$$

The parameter μ is equivalent to the trust region parameter in the Levenberg-Marquardt algorithm. In our case this is simply a guard parameter which prevents the learning rate from becoming too large. The parameter λ is a separate learning rate which must decrease based on some schedule in order to dampen exploration of the parameter space as training progresses. The term h_{kk} is the Gauss-Newton diagonal approximation to the Hessian matrix and must be summed over all connections which share the particular weight $w_{i,j}$ (with V_k the same as before):

$$h_{kk} = \sum_{(i,j)\in V_k} \frac{\partial^2 J}{\partial u_{i,j}^2} \qquad \text{Equation 27}$$

A back-propagation implementation can be tested for correctness by using symmetrical central differences, which is a more accurate version of the finite differences method. By perturbing each weight in turn by some ϵ where $\epsilon << 1$ we can approximate the derivative of the error function by using:

$$\frac{\partial J}{\partial w_{j,k}} = \frac{J(w_{j,k} + \epsilon) - J(w_{j,k} - \epsilon)}{2\epsilon} + O(\epsilon^2) \qquad \text{Equation 28}$$

The cost of this method is the computational steps required to perform the estimation. In the case we have W weights and biases in the network; we must perform 2W or $O(W)$ forward-propagations. This in turn needs to be completed for every single weight in the network which means the overall complexity is $O(W^2)$.

2.4.2 Evaluation

As mentioned in the previous section MSE was used as the performance evaluation metric to drive network training. However MSE is a difficult metric to use if one wishes to determine how the network is performing in terms of classification since, in general, MSE does not necessarily reflect the actual classification performance of a network since it's possible to have lower MSE but have worse classification performance. Since we only have 2 possible classes (sky and ground) for each pixel at the network output, we can use a confusion matrix to summarize network performance. If we consider sky pixels as belonging to the positive class and ground pixels to be from the negative class our confusion matrix is as shown in Table 1. This allows us to ask questions such as how accurately the network is classifying pixels overall? Or what percentage of actual sky pixels we see show up as ground pixels (false negative rate)?

	Predicted Sky	Predicted Ground
Target Sky (Positive)	True Positive Count (TP)	False Negative Count (FN)
Target Ground (Negative)	False Positive Count (FP)	True Negative Count (TN)

Table 1 - Confusion matrix for summarizing the classification results on a sky-mask output generated by a network

Since neurons in the output layer of a CNN are bounded (saturating) in their output between a minimum and maximum value we must select an appropriate threshold to use in order to classify

whether the particular neuron has detected ground or sky at its location. Since the activation function for neurons in this thesis is the bipolar and symmetric hyperbolic tangent function we can select any value in its output range as the threshold, choosing a sky label in cases where a neuron generates activation greater than or equal to the threshold and ground otherwise. The special case of selecting 0 for the threshold is defined as accuracy and represents the case where we assign equal significance or cost to incorrect classifications. After selecting a threshold we can tabulate a confusion matrix based on the classifications of image pixels as sky or ground as generated by a trained network. The confusion matrix can be accumulated for either a single image or an entire training or test set.

In order to evaluate the training performance on horizon-mask networks, accuracy will be useless since the horizon-mask will not necessarily be the horizon boundary; this was shown in Figure 6. Therefore if noise resiliency is desirable, or as will be discussed to even allow the networks to train successfully as horizon-mask detectors, a more specialized metric is necessary. To achieve this, we evaluate how much of the ground truth target horizon is captured by the generated output from a horizon-mask network. If the number of pixels capturing the horizon boundary is N, then applying the logical-AND between the output of a horizon-mask network and the target output should produce a number of pixels M. The ratio M/N then gives the classification success rate and can be averaged over a training or test set.

2.5 Stereoscopic Vision

This section provides a background to stereo vision and outlines the approach taken in the remainder of the thesis in order to process a stereo pair of images of a scene, extract depth information to potential obstacles and to create a camera centered model of the obstacles in 3D which can then be passed to an obstacle avoidance algorithm.

2.5.1 Camera Model

The pinhole camera model, shown in Figure 10 below, is a model of an idealized pinhole camera used to simplify the underlying mathematics. A pinhole camera assumes a small light permitting aperture which allows rays to enter a camera and reach a retinal plane while allowing correspondence between object and image points to be retained. By using a pinhole camera model, simple equations can be used to describe the perspective projection which occurs of 3D objects onto the 2D retinal plane. Assuming (X,Y,Z) to be the coordinates of a point in 3D space, we can project point (x, y, z) on the retinal plane, given a focal length f and center of projection O. The retinal plane is a distance f from the center of projection along the optical axis:

$x = f\frac{X}{Z}$ Equation 29 $y = f\frac{Y}{Z}$ Equation 30 z=f Equation 31

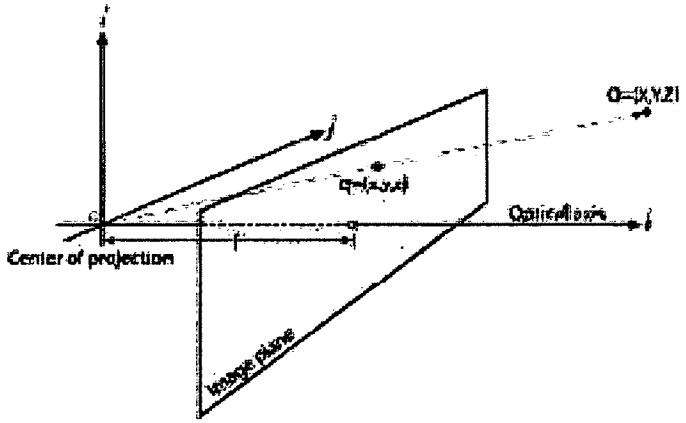

Figure 10 - Pinhole camera model [82]

Since the points (x,y) are measured in the camera reference frame (CRF) they also have physical units. Since images features are typically referred to by their pixel location a conversion must be made from the CRF to image coordinates x_{img} and y_{img}. The conversion can be written as:

$$x_{img} = s_x * x + c$$

$$y_{img} = s_y * y + c_y$$

The parameters s_x and s_y are the horizontal and vertical scaling factors respectively and c_x and c_y the principal points. The scaling factors describe the physical dimensioning of pixels while the principal points describe the origin offset of the camera reference frame with respect to the retinal plane. The retinal plane is typically constructed as an array of light sensitive elements which convert light energy into an electric field with the sensor typically being either a Charged Coupled Device (CCD) or Complimentary Metal-Oxide Semiconductor (CMOS). These sensors typically vary in quality, cost, power consumption and noise resilience with each being better suited in certain applications. CMOS is typically prevalent in low end devices suited for mass consumption such as webcams due to lower manufacturing costs.

Selecting an appropriate camera with the right sensor is an important task for any vision system which must deal with changing conditions. Considerations such as resolution, sensitivity, field of view, noise resilience and cost to name a few must be taken into account. For example, selecting a camera for an aircraft whose sensor yields a large amount of image corruption due to mechanical vibration can lead to problems. These problems can be compounded when a stereo system with multiple cameras is required, since conditions will rarely be perfect. Problems can include bad or changing illumination or differences in camera construction.

2.5.2 Camera Calibration

The calibration process allows us to determine camera parameters. This involves determining the internal camera parameters, known as the intrinsic parameters, as well as the external or extrinsic parameters. The intrinsic parameters include its focal length, principal points and distortion factors. The focal length is the distance from the center of projection to the retinal plane while a

principal point indicates the offset of the retinal plane relative to the center of projection of a camera. Distortion factors on the other hand arise as a result of imperfect lens construction, distorting the image geometry. Typically only radial distortion is taken into account due to its noticeable effect of image stretching, however tangential distortion which arises from improper lens alignment and skew distortion which occurs when retinal plane axes are not orthogonal may also be considered.

Extrinsic parameters on the other hand, describe the positioning and orientation of the camera reference frame (CRF) relative to some known world reference frame (WRF). This means that the extrinsic parameters describe how to translate and rotate 3D points in the WRF into 3D points in the CRF; typically performed using a single homogeneous matrix transformation. A homogeneous matrix transformation is used in order to linearize the originally non-linear perspective camera model. If we take the intrinsic parameter matrix M_{int} and extrinsic parameter matrix M_{ext}, we can write the transformation of a 3D point (X,Y,Z) in the WRF in homogeneous coordinates as:

$$\begin{bmatrix} x_{img} \\ y_{img} \\ 1 \end{bmatrix} = M_{int} M_{ext} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Equation 32

$$\begin{bmatrix} x_{img} \\ y_{img} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x * s_x & 0 & c_x \\ 0 & f_y * s_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The rotation parameters r_{ij} and translation parameters T_x, T_y & T_z in the extrinsic matrix are precisely the parameters required for transformations between WRF and CRF while the intrinsic matrix projects the rotated and translated point in the CRF onto the retinal plane. The parameters

f_x and f_y are simply the focal lengths of the camera in their respective directions which are not necessarily equal. The parameters s_x and s_y are scaling factors dependent on the construction of the camera imaging array and give the conversion between physical and pixel units; these also need not be equal since pixels are not necessarily square. In the case where camera distortion is taken into account, the distortion parameters can be applied to the coordinates to determine the correct locations of feature points in the image.

In order to determine the unknown parameters in the intrinsic and extrinsic matrices as well as the camera distortion parameters, one typically acquires images of a reference pattern with a known geometry and then uses one of a number of calibration algorithms to fit the parameters [81]. An example of this process is shown in Figure 11 where the user presents a checkerboard pattern to a camera, capturing images of the board at different orientations. The calibration problem then becomes one of determining the parameters for a planar homography which maps points on the 2D checkerboard plane to the 2D image plane of the camera [82]. Having a number of views of the reference object (as well as a large, easily discernible object) not only helps to fulfill any constraints of the calibration algorithm but also provides some resilience to image sampling noise and numerical instability.

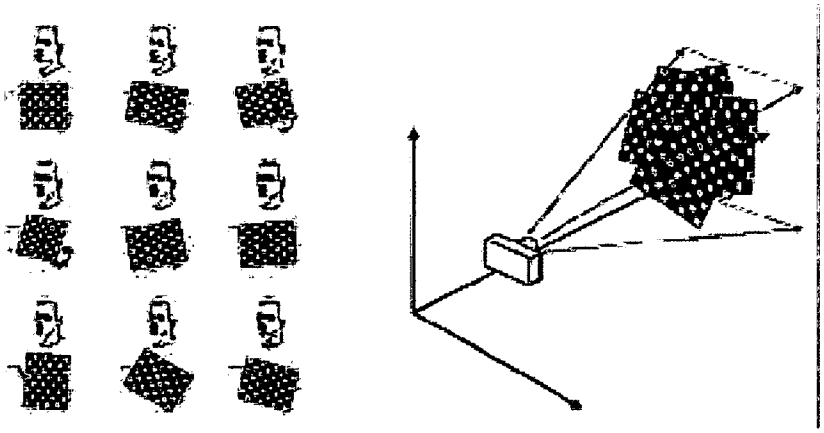

Figure 11 - Single camera calibration process [82]

The calibration process is similar for calibrating a pair of stereo cameras and typically the intrinsic parameters of each individual camera need to be determined anyways. However the rotation and translation parameters acquired from the calibration are then one camera relative to the other. In addition to this, image pairs coming from both cameras are needed of the reference object. In this case of stereo cameras, determining the camera parameters is the first step before stereo calibration is performed. Stereo calibration allows for the setup of the camera system to be determined using epipolar geometry, allowing for rectification of stereo image pairs.

In the case of UAVs, and depending on the mounting of the cameras, there may be a problem of a changing calibration over the duration of a test flight. This can occur due to vibration or flexing of the mounting surface or camera mount. Assuming that the intrinsic parameters of the camera are fixed (the focal length remains the same), auto-calibration methods do exist for using uncalibrated images (images having no a-priori calibration objects) [79] which rely on known point correspondences between both views. Although deploying such a system could be beneficial for the GeoSurv in the future, the assumption made in this thesis is that the calibration parameters remain fixed for particular flights and ground tests. In the case where this assumption is violated,

the correspondence stage of the processing pipeline could cease to produce a coherent result which means that firmly fixing the cameras relative to each other is currently paramount.

2.5.3 Stereo Rectification

Stereo images are typically rectified in order to simplify the process of determining depth information from the images. The rectification process mathematically aligns the stereo images so that they appear that they were taken with 2 cameras whose image planes are row aligned; a frontal parallel configuration. This mathematical alignment is preferable to physically aligning a pair of cameras which is a next to impossible task.

To better define the basic geometry of a stereo system, we refer to what is commonly known as epipolar geometry [81]. Consider the configuration shown in Figure 12 where 2 pinhole cameras and their respective retinal planes are shown. The optical center O_L of the left camera is located at a focal length of f_L from the left retinal plane P_L. A similar configuration follows for the right pinhole camera. The optical centers of the cameras are separated by a baseline T. If we consider a 3D point p in the field of view of both retinal planes, we can then visualize a triangle with vertices O_L, p and O_R. The edges of the triangle are then 2 lines, O_{Lp} and O_{Rp}, extending from the left and right optical centers respectively to the point p. The remaining edge is the baseline between the 2 cameras. The line O_{Lp} intersects the left retinal plane at the point m_L, corresponding to point p as seen by the left pinhole camera. A similar observation can be made for the right pinhole camera and the point m_R. Finally, the baseline connecting the optical centers of the cameras also intersects both retinal planes at special points called the left and right epipoles, shown as e_L and e_R respectively. These points correspond to the optical center of the left camera as seen on the retinal plane of the right camera and vice versa.

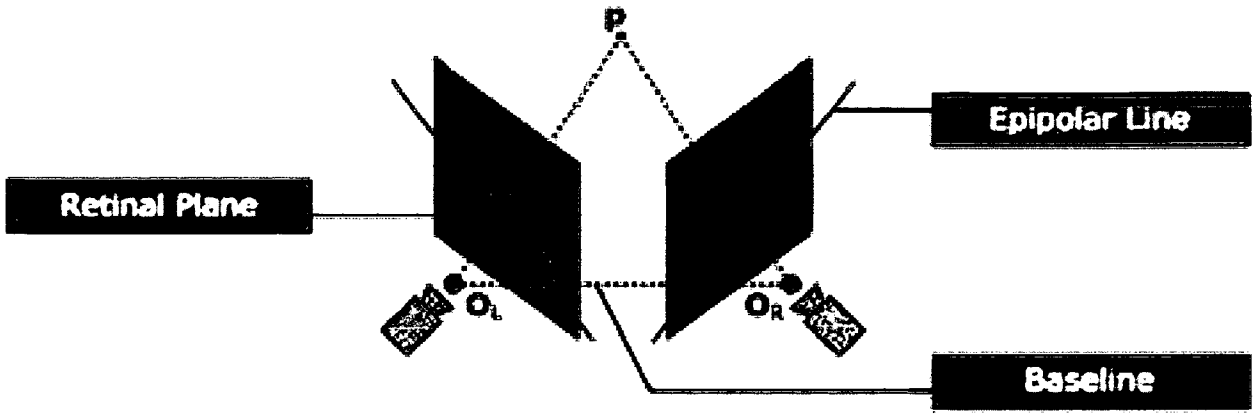

Figure 12 - Epipolar geometry of a stereo pair of pinhole cameras

Since only 3 points are needed for a unique plane in 3D, we can extend the triangle into an epipolar plane which intersects the 2 retinal planes. When we do this, we form a line called the epipolar line on each retinal plane. We then see from Figure 12 by construction, the epipolar line must lie on the epipolar plane and an epipolar line must pass through the projected point m_L or m_R and the corresponding epipole for the left or right camera. Further to this, each epipolar line represents the view on the retinal plane of the line from the optical center to point p for the other pinhole camera. Two significant observations can now be made about this view. The first is that any 3D point p in the field of view of both cameras must follow this geometry. The second is that having the coordinates of a point on either retinal plane reduces the search for the location of the corresponding point on the other retinal plane into a 1D search along an epipolar line (known as the epipolar constraint).

By creating an epipolar constraint, we simplify the search for matching points on our retinal planes. However, having to explicitly determine each epipolar line for all point pairs is a cumbersome process which can be simplified. The simplification comes as a result of our assumption of a pinhole camera model by applying a special kind of homography called a perspective trans-

formation [81]. By using a perspective transformation, we are able to relate a pair of images of an object taken by 2 pinhole cameras with another pair of images having a different projection of the same object. By selecting the proper transformation, we can make our epipolar lines parallel and horizontal using a 3x3 homography matrix.

The result of a rectification of a pair of stereo images is shown in Figure 13 below. Note how features in the original image pair (top) are not aligned; however after rectification (bottom), the resulting epipoles are at infinity and the epipolar lines now align corresponding pixels horizontally. There are a number of approaches to computing the rectification, among them being Hartley's algorithm [102] and Bouguet's algorithm [103]. The former is an un-calibrated stereo rectification algorithm which assumes only a fundamental matrix. The latter method, which we use via the MATLAB Vision Toolbox, is for calibrated stereo and assumes the intrinsic parameters of the stereo cameras are known and which we determined through calibration. Having completed rectification of a stereo pair of images, the matching of feature points along corresponding epipolar lines in the 2 images can be performed.

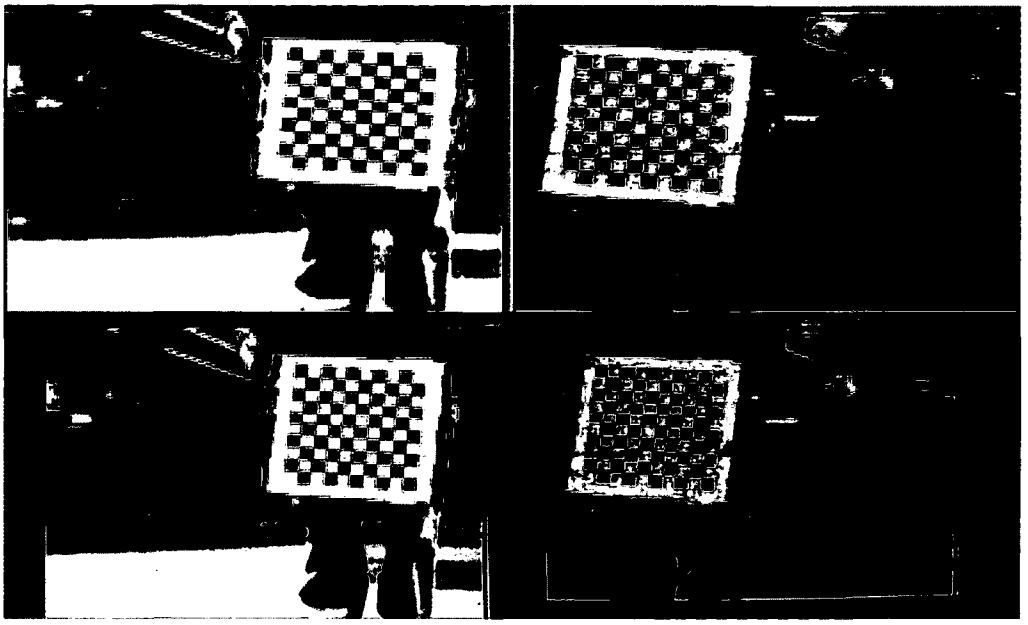

Figure 13 - Original image pair (top) and stereo rectification result (bottom)

2.5.4 Stereo Correspondence

Stereo correspondence is the process by which matching pixels between separate images of a scene taken from different cameras are found. Since a point in an image could potentially match many points in another image epipolar geometry constraints and image rectification are widely used to simplify the process, along with other constraints, allowing the search area for matching pixels to be narrowed and thus be more computationally efficient.

The type of stereo correspondence performed typically falls into 2 categories: sparse or dense stereo. Sparse stereo matching is useful when a small amount of depth information is sufficient and computational resources are at a premium. In order to extract features for matching, feature extraction methods such as SIFT, SURF or FAST can be used or a simple approach such as edge detection [83][84][85]. Although sparse stereo methods by definition provide less depth information than their dense counterparts by focusing on pixels or features, seed-and-grow methods

can be utilized to improve results [63]. On the other hand dense stereo methods, attempt to extract depth information over as much area as possible. While offering more depth information, these methods can be computationally intensive [64].

In general, stereo correspondence methods face a number of issues that must be resolved in order for the algorithm to be effective. Stereo images with wide baselines have the advantage of a higher disparity resolution, that is, how fine grained the depth perception is (this effect is described in Section 2.5.5 in more detail); however they are also burdened with a larger number of occluded areas. An occlusion is a region in a stereo image which is not visible in the other image due to the positioning of the cameras or an object blocking the view. A further source of occlusions comes as a result of 2 stereo cameras having different fields of view, with one camera seeing a portion of the scene that the other cannot. A narrow baseline helps to reduce occlusions since both cameras view the scene from similar perspectives, however the disparity resolution decreases. Regions of low texture are also problematic proving little information with which to perform correspondence.

The work of Scharstein and Szeliski in [64] outlines a taxonomy, or essential building blocks, for dense stereo correspondence algorithms; dense stereo algorithms typically perform some subset of these building blocks. The buildings blocks can be summarized as follows:

1. Matching cost computation
2. Cost aggregation
3. Disparity computation and optimization
4. Disparity refinement

Dense stereo algorithms typically break down into local and global approaches [64][86]. Local dense stereo algorithms typically rely on knowledge of a small part of the image and perform matching cost computation between the left and right images using a support region such as Sum of Squared Differences (SSD), Sum of Absolute Differences (SAD) or Normalized Cross Correlation (NCC). Other more complicated similarity measures which take into account potential variation in sampling between both cameras have also been developed such as the work in [78]. In the process of matching, local dense stereo algorithms typically make certain assumptions in terms of the commonly used constraints mentioned above such as assuming smoothness in disparity estimates by aggregating matching costs over a localized region.

More complicated global dense stereo algorithms are also common which typically define an explicit global cost function (over the entire image) which encompasses matching costs while explicitly defining a corresponding smoothness penalty. The cost function is then minimized in order to find the optimal disparity assignment using one of a number of optimization techniques, including graph cuts [87] and belief propagation [88] among others. Although accurate, a common pitfall of these approaches is their computational performance which can be slow even when implemented on a GPU. As mentioned since the general global optimization problem is NP-hard simplifications of the problem are warranted; a common approach being dynamic programming. This approach works by decomposing the difficult 2D problem into a number of 1D problems, performing the "global" optimization on individual pixel rows (scan-lines) [89]. As this approach can lead to misalignment between scan-lines, more elaborate approaches such as assuming a tree structure across the image are possible [90] which retains a mixture of vertical and horizontal adjacencies between pixels, allowing horizontal and vertical smoothness constraints to be retained.

Finally, the last step in the list of building blocks is disparity refinement which, for example, may perform sub-pixel refinement from the originally computed disparities in order to reduce the uncertainty in range estimation [91].

2.5.4.1 Feature Extraction for Sparse Stereo Correspondence

In order to extract features for sparse stereo correspondence edge detectors are used. A number of operators or methods for performing edge detection are common including Sobel, Prewitt and Canny [92]. The Sobel operator is two 3x3 mask windows (one for horizontal and one for vertical edges) which are convolved with the original image and produce an approximation of the image intensity gradient at each image pixel; both masks are shown in Equation 33. Regions with large gradients (large changes of intensity) are then candidates for edges. A threshold can be applied to computed gradients to remove edges which are likely to end up being noise. The Sobel operator places more emphasis on pixels closer to the center pixel (signified by the +2/-2 scaling factors) compared to the Prewitt operator, which has almost the same masks except that no emphasis is placed on pixels close to the center pixel. Other operators are also possible, such as the Roberts operator which emphasizes edges on diagonal lines. An example of applying the Sobel operator (threshold of 8) on an image from a flight video is shown in Figure 14.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad \text{Equation 33}$$

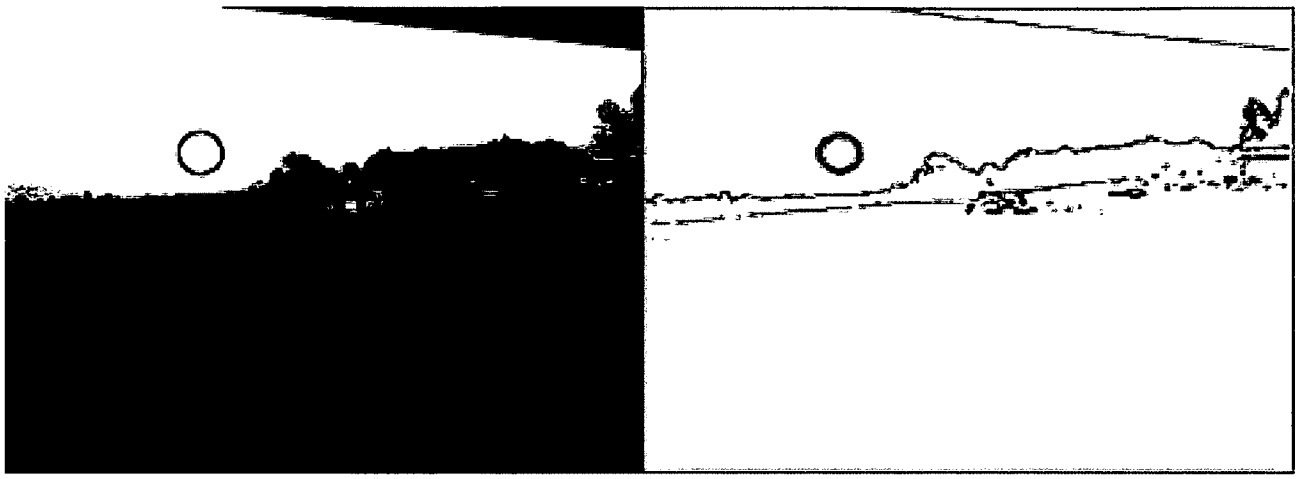

Figure 14 – Image from Telemaster flight at the Arnprior field (left) and the corresponding Sobel edge map (threshold 8, right). Balloon obstacles are circled in red for clarity. Image is inverted and highlighted in red for clarity.

2.5.4.2 Sparse Feature Block Matching

Having computed locations of interest for matching in the image using edge detection, stereo correspondence must occur in order to obtain a depth map. To perform stereo correspondence of edge maps we could potentially attempt 2 solutions:

1. Mask the original grayscale or color images using the corresponding edge map to highlight the relevant image features and perform stereo correspondence on the resulting "color-mask" images.
2. Use the edge map to focus the correspondence by only performing correspondence at locations where an edge was detected. The correspondence itself however would not be performed on the "color-mask" images but on the originals.

Figure 15 highlights the first point, where the original image pair is processed using the Sobel operator (threshold of 32) which is then used to mask out color details not found in the edge map to obtain color-mask representations of the originals. Correspondence can then proceed between both color-mask images. This has the advantage of significantly focusing the image detail so that in places where no color information is present (in the color-mask), no correspondence needs to be performed. However it comes at a price since the amount of detail contained in any matching window will be greatly reduced as well, so that matching horizontal regions along the horizon

could lead to spurious results. The alternative is described in the second point where the matching process is performed only at locations where edge information is present; however the matching windows are applied to the original images instead of the color-mask images. The goal with this approach is to reduce matching noise, while still reducing computational complexity.

In the case of stereo images, a left and right disparity image can be computed by using the left or right image as the reference. Since each image is distinct, the correspondence process from left to right and from right to left will not produce identical results. A consistency check can then be performed to ensure that both the left and right disparity maps agree on the selected disparity for a pixel. The level of disagreement allowed between both disparity maps for a pixel is based on the application; however we adopt a hard difference threshold of 0 to attempt to obtain as accurate results as possible. If we consider approach #1 above to sparse matching, we notice that the detail available for matching is only edges. This could circumvent the consistency check since a particular feature may not show up as an edge in both the left and right images but still end up being matched consistently between both images (since it may be the only edge within the searched disparity range). On the other hand approach #2 will assign disparities having searched the entire disparity range around a pixel, making the task of consistency more difficult since matches are possible at all disparity values (not just those at which edge information is available). For this reason we focus on approach #2 for performing feature matching in this thesis.

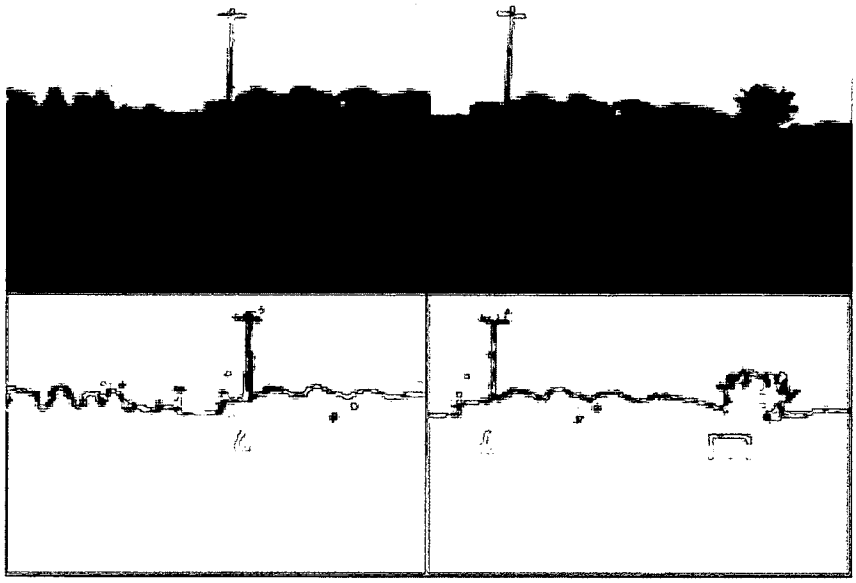

Figure 15 – Left and right stereo image pair (top row) and corresponding color-mask images obtained from applying a Sobel edge mask to each (bottom row). Edge map is inverted and highlighted in red for clarity.

In order to perform the block matching a number of area-based metrics are possible. This includes sum of absolute differences (SAD), sum of squared differences (SSD) or normalized cross correlation (NCC). SAD (Equation 34) and SSD (Equation 36) are the simplest computationally, calculating the pixel wise differences between two small windows located in each respective image. However, these metrics are sensitive to radiometric distortions which can arise when the pixel intensities in one image differ by a constant offset and/or gain factor to the corresponding pixels in the corresponding stereo image [76]. If we take W to be an NxN (N is odd) window where the pixel locations (u,v) inside the window are 0-based and relative to the top left position (x,y) of the window inside the image then for a disparity d, between the left image I_1 and right image I_2, the SAD and SSD measures can be written as:

$$\sum_{(u,v)\in W} |I_1(x+u,y+v) - I_2(x+d+u,y+v)| \qquad \text{Equation 34}$$

$$\sum_{(u,v)\in W} (I_1(x+u,y+v) - I_2(x+d+u,y+v))^2 \qquad \text{Equation 35}$$

NCC is able to deal with the problem of a gain factor and its variants, such as zero normalized cross correlation (ZNCC), are able to take into account the problem of a constant offset as well [77]. Taking the same definitions as those for the SAD and SSD measure and also taking $\overline{I_1}$ and $\overline{I_2}$ to be the mean of the window W in image I_1 and I_2 respectively, the ZNCC measure is:

$$\frac{\sum_{(u,v)\in W}(I_1(x+u,y+v) - \overline{I_1}) * (I_2(x+d+u,y+v) - \overline{I_2})}{\sqrt{\sum_{(u,v)\in W}(I_1(x+u,y+v) - \overline{I_1})^2 * \sum_{(u,v)\in W}(I_2(x+d+u,y+v) - \overline{I_2})^2}} \qquad \text{Equation 36}$$

Finally, the Birchfield and Tomassi (BT) metric will also be investigated since it is pixel matching metric which is insensitive to image sampling [78]. This metric can be used to match single pixels between images or can be used as part of a window by accumulating the matching costs for all pixels in a window.

2.5.5 Disparity Computation and Depth Re-projection

Having calibrated the stereo cameras, rectified a stereo image pair and computed the stereo correspondence of an image pixel, we can triangulate the physical distance to the 3D point corresponding to the pixel. Assuming a frontal parallel pinhole camera arrangement or fully rectified stereo images we can view the triangulation as shown in Figure 16 below. By using the knowledge of the baseline, T, between the optical centers of the pinhole cameras, their focal

length and the pixels location x^l and x^r in the left and right images respectively, we can use similar triangles to determine the depth parameter z. As a simplification we assume that the focal lengths, f, of the cameras and image widths, w, are identical.

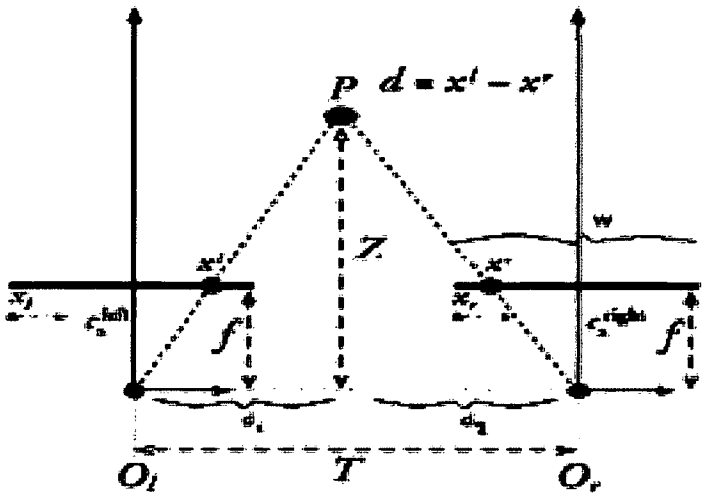

Figure 16 - Frontal parallel image pair pixel triangulation [82]

For the left retinal plane we have a ratio of similar triangles, where the left inner triangle is formed by the pixels coordinate, x^l, relative to the center of the screen and the focal length f while the left outer triangle is formed by d_1 and the depth z to point P:

$$\frac{x^l - \frac{w}{2}}{f} = \frac{d_1}{z} \qquad \text{Equation 37}$$

Where we have assumed d_1 and d_2 are the left and right parts of the triangle base and that:

$$d_1 + d_2 = T \qquad \text{Equation 38}$$

The right retinal plane gives a right similar triangle with the triangle base between the right screen pixel, x^r, and the center of the right retinal plane:

$$\frac{\frac{w}{2} - x^r}{f} = \frac{d_2}{z} \qquad \text{Equation 39}$$

Rearranging Equation 37 and Equation 39 and substituting into Equation 38 gives:

$$\frac{z}{f}\left(x^l - \frac{w}{2} + \frac{w}{2} - x^r\right) = T$$

$$z = \frac{fT}{x^l - x^r} = \frac{fT}{d} \qquad \text{Equation 40}$$

The variable **z** is a product of the focal length measured in pixels, the baseline measured in physical units and the disparity $d=x^l - x^r$ which relates to the difference in horizontal pixel position between the left and right images. This simple depth equation can be modified to incorporate different focal length between cameras as well as the horizontal principal points of the cameras in the case where the camera principal rays do not intersect at infinity. We can immediately see that due to the inverse relationship between depth and disparity, disparity decreases as depth increases. In Figure 17 below, we show lines of constant disparity for a pair of cameras. Note the non-linear increase in depth resolution due to a linear decrease in disparity.

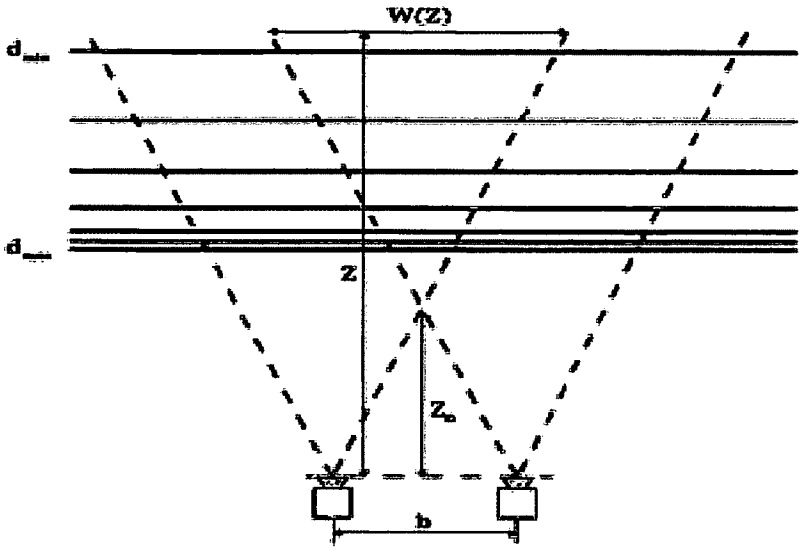

Figure 17 - Planes of constant disparity for a stereo camera pair [82]

Having obtained a disparity for the pixels in our image, we need to be able to re-project the pixels into 3D space if we hope to build a model for obstacle avoidance. To accomplish this, we can use the re-projection matrix Q.

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & 1/T & (c_x - c_x')/T \end{bmatrix} \qquad \text{Equation 41}$$

This matrix is a compact version of the camera reference frame centered 3D to 2D projection equations described in Section 2.5.1 and all parameters can be acquired through camera calibration. It can be modified to take into account different focal lengths in the horizon and vertical directions (a single focal length f is given in Equation 41). The terms c_x and c_y are the principal point coordinates for the reference image (left or right). The term c_x' is the horizontal coordinate of the principal point in the opposite image. In the case where the principal points are set to intersect at infinity, we have $c_x = c_x'$. The re-projection matrix allows us to compute physical 3D coordinates to feature points using the following expression:

$$\begin{bmatrix} X_h \\ Y_h \\ Z_h \\ W_h \end{bmatrix} = Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix}$$

Where X_h, Y_h, Z_h and W_h are the homogeneous coordinates of a pixel having image coordinates (x,y) and disparity d. The 3D coordinates (X, Y, Z) of the feature point can then be recovered since:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_h/W_h \\ Y_h/W_h \\ Z_h/W_h \end{bmatrix}$$

Being able to compute the depth to pixels we need to take into account 2 limitations to any stereoscopic camera system: range accuracy and range resolution. Range accuracy is a measure of how close our estimate of disparity or depth is to the ground truth depth for a particular 3D point. This accuracy is affected by a number of factors, including errors introduced during the calibration process as well as errors in the stereo correspondence process. Range resolution on the other hand is a fundamental limitation of a stereoscopic system at discerning how finely range measurements will vary at a particular depth, for a given pixel or feature point, based on variations in the disparity estimate. To compute range resolution Δz, we can consider Δd to be the corresponding disparity delta which then allows us to write the following expression for Δz:

$$\Delta z = \frac{T * f}{d} - \frac{T * f}{d + \Delta d}$$

$$= \frac{Tf}{d} \frac{\Delta d}{d + \Delta d} \qquad \text{Equation 42}$$

$$= \frac{\Delta d * z^2}{T * f + z * \Delta d} \qquad \text{Equation 43}$$

where Equation 42 simplifies to Equation 43 by using Equation 40 twice to substitute for $\frac{Tf}{d}$ and for d. If we now take a first order Taylor expansion about $\Delta d = 0$ we obtain:

$$\Delta z \approx \frac{z^2}{f * T} \Delta d \qquad \text{Equation 44}$$

This measure is a direct consequence of the inverse correspondence between range and disparity and represents the theoretical best range resolution for a stereo system. We see that for a particular stereo system (fixed f and T), our resolution error squares with the depth to the feature points we're considering. This relationship is a physical limitation and in practice it is likely that the true resolution will be even lower due to limitations on our range accuracy. To obtain better range resolution we can make a few choices in our selection of cameras; using a wider camera baseline, longer focal length or a higher resolution sensor. This in turn will affect the difference between the camera viewpoints (larger variation for increasing baseline), the depth at which the views overlap (further for increasing baselines), the field of view of our system (smaller for increasing focal length) and the processing time for correspondence (higher for larger resolution images). In addition, we can introduce sub-pixel interpolation to our stereo correspondence process which will reduce the minimum size of Δd. For example, instead of resolving disparities to a difference of only a single pixel we can interpolate to ½ or ¼ pixels giving Δd values of ½ or ¼ respectively. The increased resolution however comes at the cost of increasing computational requirements.

To illustrate we can consider the stereo cameras which were mounted on the Telemaster for flight tests. The stereo calibration for these cameras using the MATLAB toolbox produces a focal length of 607.366 pixels and baseline of 1.292 meters (true measured baseline of 1.32 me-

ters). From these calibration values we can plot range resolution curves using 4 disparity resolutions of 1, 1/2, 1/4 and 1/8 as shown in Figure 18 below.

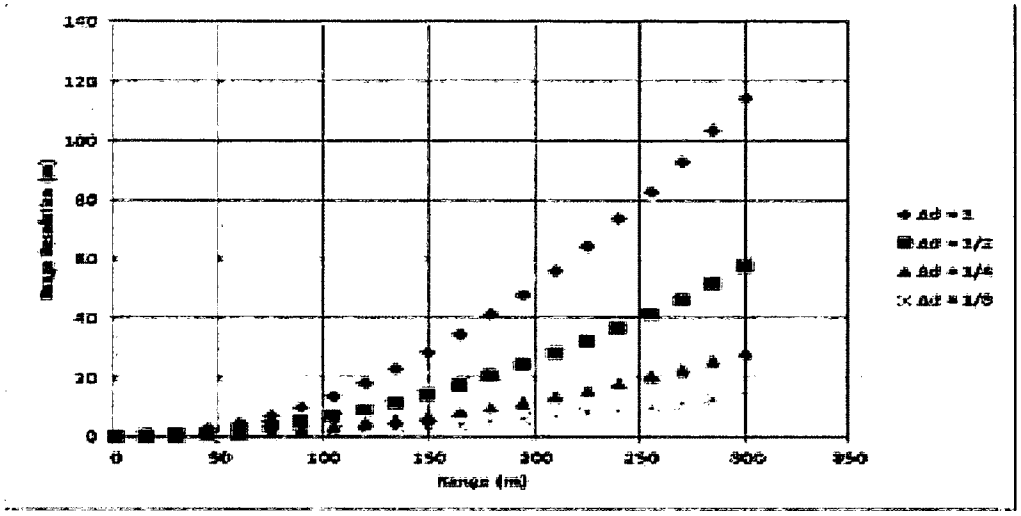

Figure 18 - Plot of range resolution (Δz) versus the physical distance to a pixel (z) for different disparity resolutions

We can see that in the case of a non-interpolation approach, range measurements beyond 60m have over 4.5m of uncertainty in their estimation. Moving to an interpolation approach, say 1/8, allows us to measure up to 170m before we obtain the same degradation in resolution. In addition to the considerations mentioned above, the resolution which is usable will also depend on the size of the obstacles which are expected, the current speed of the aircraft and the processing speed of the stereo correspondence algorithm. In addition, the obstacle avoidance portion may not require range resolution below a certain point in the avoidance calculations. As explained in Section 2.2.1, a bounding box region will be placed around potential obstacles by the avoidance algorithm. In the case where the size of the bounding box is larger than the depth resolution, at the maximum depth one expects to see, it may not be worthwhile to invest too much effort in adjusting parameters to reduce the range resolution.

Although having the re-projection matrix allows one to estimate the maximum perceivable distance for the stereo system (using the smallest discernible disparity value), one must still be mindful of the resolution limitations in the camera system relative to the smallest obstacles which are expected to be encountered. For example if we consider Equation 41 and take the Telemaster mounted camera calibration results which produced a focal length of 607.366 pixels, baseline of 1.292 meters and principal points c_x=228.034 and c_y=151.307 then for a pixel at the center of the image and at a minimum disparity d=1 we find a maximum depth of 785 meters. To get an idea of the size an obstacle would need to be in order to register at this distance we can construct the following diagram:

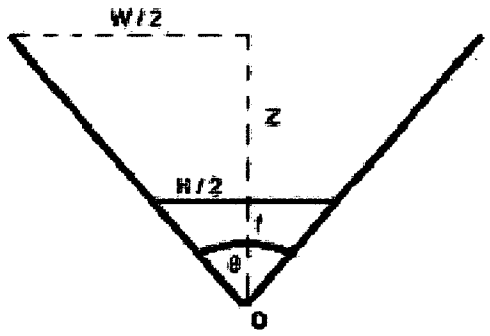

Figure 19 – Construction for estimating the physical dimensions subtended by a cameras field of view

In Figure 19 we have the image plane (grey line) having a center of projection O at a distance of f with a dimension of H pixels. At some distance Z from the camera the angle of view spans a distance W in physical units. The angle of view in the horizontal direction (the view is top-down) is θ. Thus we have:

$$\frac{W}{2Z} = \tan(\frac{\theta}{2})$$

If we then multiply through by 2Z and divide by the horizontal sensor dimension H (in pixels), we have:

$$\frac{W}{H} = \frac{2Z \tan(\frac{\theta}{2})}{H}$$

With an angle of view of 50°, at a maximum distance of Z=785 meters and a sensor dimension of H=640 pixels we have W/H=1.14 meters/pixel. Although this construction is an idealization, it gives us a rough estimate that an object should have high enough contrast and be at least this size horizontally or risk not being detected by the imager. A similar argument can be applied to the vertical angle of view.

The disparity results obtained must first be grouped to obtain bounding regions for nearby pixels having similar depth. This grouping is performed by using the OpenCV implementation of the connected component algorithm [101] which aggregates adjacent pixels having some similarity. Similarity is taken to be any pixel which is an 8 connected neighbor to other pixels with the same disparity as shown in Figure 20; 2 pixels located on the same disparity plane (are both at the same depth) which are also 8 connected neighbors are considered as candidates for grouping.

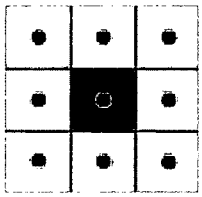

Figure 20 – Center pixel (red) and its 8 connected neighbors

The reason for this approach to pixel aggregation is foremost that the obstacle avoidance portion of the processing pipeline is expecting 2D rectangular bounding regions for obstacles which are displaced by some distance along the depth axis. Typically once bounding regions on each disparity plane have been determined, regions across disparity planes can be further aggregated; for

example at shorter depths where consecutive disparity planes are closer than a certain threshold. This processing is performed instead by the obstacle avoidance processing stage which has its own requirements, parameter settings and algorithm for performing aggregation of bounding regions before any avoidance computations are performed [80]. For example, the current avoidance algorithm assumes that any 2D region on a disparity plane is projected as a rectangular prism into 3D space by using the physical width of the bounding region as its depth. All rectangular prisms are also expanded in volume by a heuristic factor which takes into account the avoidance dynamics of the aircraft (for example minimum turning radius) and distance to the obstacle. This is done to ensure that the avoidance algorithm can actually generate enough command input to deviate from the original flight trajectory. In addition, further geometric constructions are appended to the 3D bounding region in order to simplify avoidance calculations. Finally, the only avoidance scenarios currently under investigation are 'fly-over' and 'fly-around', meaning that the aircraft is currently not expected to fly under obstacles. To incorporate knowledge of this constraint in the algorithm the reading from the GeoSurv AGL sensor is used to elongate any detected obstacle in the vertical direction, making it appear to be sticking out from the ground.

2.6 Compute Unified Device Architecture

The compute unified device architecture, or CUDA [93], is a general purpose parallel computing architecture and parallel programming model developed by NVIDIA. It gives developers access to the computing elements in NVIDIA graphical processing units (GPU's), allowing them to be used for general purpose applications such as stereo vision. In doing so, it provides an attractive alternative over traditional CPU based computing by offering a general purpose, low cost, computationally powerful computing platform.

At the program level, CUDA exposes the underlying computing power of the GPU through threads which are lightweight units of execution. Program code called a kernel is written to execute in a thread, possibly reading and writing data from memory and performing computations. Threads are further aggregated in a hierarchy, the bottom of which is the individual thread unit. Above this level threads are aggregated into blocks, which are groups of threads that share computational resources such as fast memory (referred to as shared memory) and registers. Thread blocks are further grouped into a grid, with different thread blocks representing independent computational units which cannot directly synchronize together. Both blocks and grids are 3 dimensional constructs, which allows programmers to create a logical configuration of thread resources depending on their application. An example of the detailed hierarchy is shown below in Figure 21.

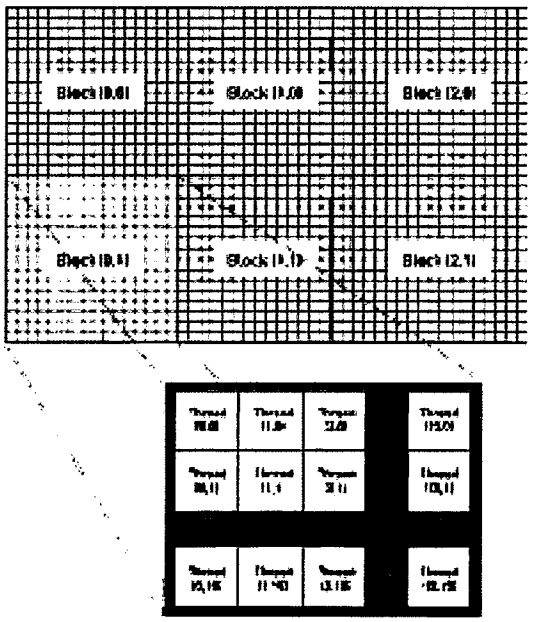

Figure 21 - CUDA thread hierarchy [93]

The underlying hardware of a CUDA GPU is arranged in lightweight streaming processors (SPs), a number of which are aggregated on a streaming multiprocessor (SM). A CUDA GPU

has a number of SM's (and therefore a number of SP cores), the number of which depends on the cards supported compute capability. Each SP has its own control logic and instruction cache and can execute a group of 32 threads (a unit known as a warp) in parallel. Each SM has a small cache (L1) to the large global memory (GDDR DRAM), while an L2 cache is also available across all SM's. As mentioned resources such as registers and shared memory which are available to the threads in a block are limited, as are the number of thread blocks which can be resident on an MP. Nonetheless, a CUDA GPU allows a developer to instantiate grids with hundreds of thousands of threads, giving rise to vast parallel computational power. At the same time, a developer needs to understand the resource requirements of their specific application since failure to do so can lead to significant performance degradation.

The CUDA platform was used to implement both the training and evaluation framework for CNNs and to implement the block matching algorithms for stereoscopic vision in this thesis. This was done since it is expected that the obstacle detection pipeline will be run on a GPU onboard the GeoSurv aircraft in the future. In the case of the stereo vision algorithms, performance of the GPU and CPU based implementations of algorithms was compared.

3 Convolutional Neural Networks for Horizon Detection

The region of interest for obstacle detection in this thesis is the horizon and the sky. In order to extract this region for further processing, the sky and horizon must first be detected. We apply CNNs to perform this detection, the first such application of these networks to this task, showing that these networks can be trained quickly to produce highly accurate results compared to a traditional classifier used for this task (SVMs).

3.1 Experimental Setup

A persistent problem with published methods which deal with the horizon detection problem is that the datasets are usually unique making it difficult to perform comparative evaluation with previous results. As such the datasets used in this chapter were mostly collected during experimental flights using the Telemaster aircraft. These datasets are broken down into 2 groups, with each group being a particular location where the Telemaster was flown. The first location (dataset 1) is Drummond's Field which is operated by the Ottawa Remote Control Club (ORCC) and is located south of Ottawa. The second is the Arnprior RC Club (ARCC) field west of Ottawa (dataset 2). Both fields share similarities in terms of the terrain around each site including tall trees and lakes; examples of each are shown in Figure 22 below. Each dataset group is further broken down into the separate days when the flights occurred. All flights in dataset 1 occurred in the mid-afternoons during the summer of 2011. All flights in dataset 2 occurred in the span of late morning to early evening during the summer of 2012. The third dataset investigated in this thesis was obtained courtesy of SGL using a miniature UAV towed through the air using a helicopter. The UAV was equipped with forward facing cameras, capturing 720x480 video at 30 frames/sec. The flight was conducted late during a winter afternoon which contributed to a significant reduction in the amount of light in the scenes. This was unfortunate since the flight path was over particularly hilled terrain. In addition, the cameras mounted on the aircraft were interlaced, which means that even scan-lines are sampled first followed by odd scan-lines (or vice-versa). This sampling introduced a large amount of distortion in the images due to the rapid swaying motion of the towed UAV meaning that it was difficult to construct a ground truth sky-mask dataset; only 213 images were obtained. An example image from this flight is shown in Figure 23.

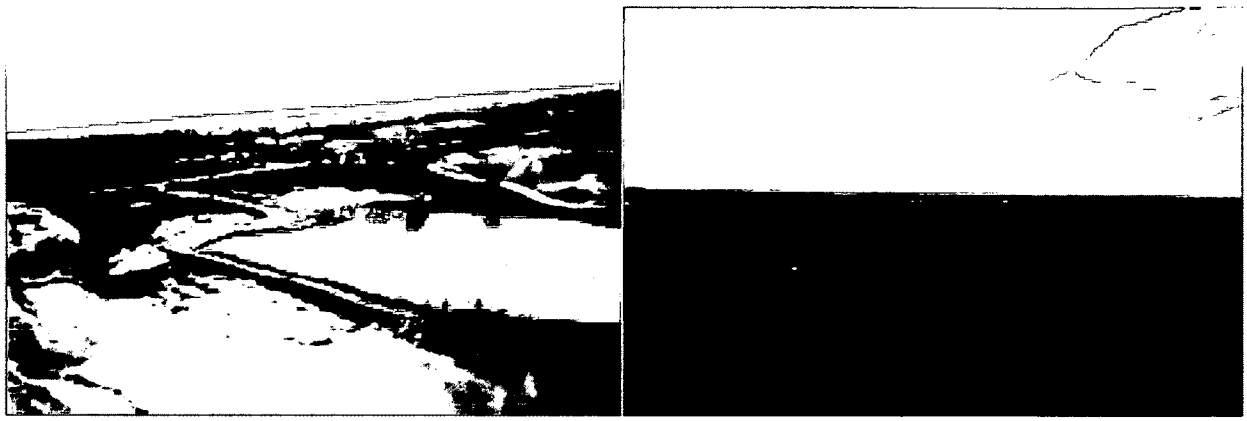

Figure 22 - Example images from dataset 1 (left) and dataset 2 (right)

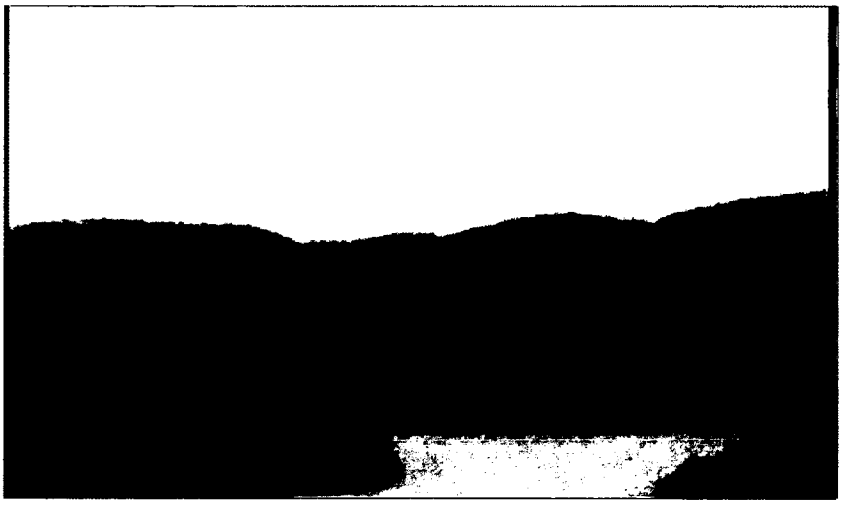

Figure 23 – Example image from dataset 3 (SGL) showing distortions introduced by an interlaced camera

The original datasets of images were processed using an existing method developed within the Carleton UAS research group [33]. The detected horizons for each image were then manually post-processed by removing false detections and a database of input RGB images and corresponding sky and horizon line images was constructed. The main reason for not continuing with the existing method is due to its low accuracy. For example, in the case of the ORCC dataset,

only 75% of all frames extracted from flight video sequences were detected as having a valid horizon. This percentage was only obtained after running the algorithm several times with a number of parameter configurations. Approximately 40% of the 75% of extracted frames had to be post-processed by hand to ensure valid horizon and sky masks could be extracted cleanly from the output. The performance of the existing method on the SGL dataset was poorer, finding the horizon in only 5% of all potential frames.

When generating ground truth horizon-mask datasets, a simple heuristic was found to work where the thickness of the target horizon-mask image is 10% of the dimension of the output image. For example for a 4 layer network with a 246x246 input layer and a 116x116 output layer, we make the horizon-mask at the output layer 12 pixels wide. Working back through the network towards the input, this dimensioning is up-sampled by a factor of 2 meaning that the ground-truth horizon mask should be 24 pixels wide. This heuristic was found to consistently lead to convergence during network training and is used to obtain the horizon-mask training results in Section 3.5. Further ground truth datasets were generated in 2 additional color spaces (YUV and Lab) in an effort to see whether the choice of color space had an effect on performance. The datasets were then either expanded, as explained in Section 3.3, or used directly to train a variety of CNN configurations.

To speed up network learning, all input image pixels were remapped from their original value range of [0, 255] to a new range of [-1, 1] using the linear transformation shown in Equation 45. There x_{in} is the original pixel value and x_{out} is the remapped pixel value.

$$x_{out} = \frac{2}{255} x_{in} - 1 \qquad \text{Equation 45}$$

The reason for this mapping is similar to the reason why weight initialization in a neural network needs to be kept to small weights; learning should proceed as quickly as possible. In the case where the inputs to a neuron are large, the neuron will undergo saturation due to its non-linear, saturating, activation function. In the saturation region, the derivative of the transfer function will be small which will make the network gradient small as well. A similar argument can be made for inputs, or a network weight initialization, which are too small. In this case network outputs will be very small, and could likewise slow down learning by producing small network gradients. Thus in addition to scaling the network inputs as described above, network weights were initialized using a uniform distribution with the following limits, where F_i is the number of fan-in input connections to neuron i:

$$[-\frac{2.4}{F_i}, \frac{2.4}{F_i}]$$

These limits are proposed in [49] with the rationale being that neurons that have a larger number of inputs should initialize themselves from a smaller weight distribution to keep their net activation large. Finally, we also follow [49] in the selection of activation function by choosing the hyperbolic tangent function (graphed in Figure 24):

$$y(net) = 1.7159 * \tanh(\frac{2}{3} * net)$$

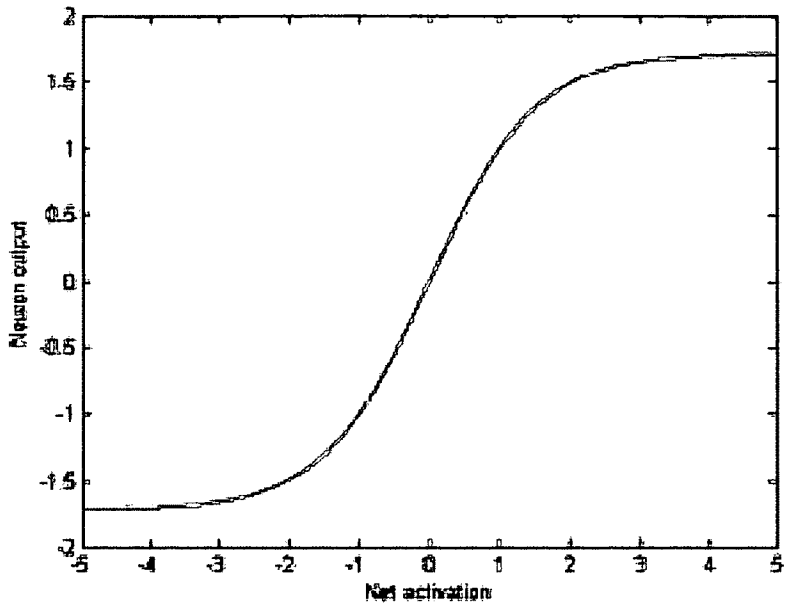

Figure 24 – Hyperbolic tangent activation function

The reasoning for choosing that activation function is that it has a wide linear operating range, with the points of maximum curvature of this function occurring around the values of -1 and 1; the normalized range of input pixels. This is not to say that the network will operate strictly within its linear region. On the contrary, we wish to restrict its initial operation to a linear region so that learning is as fast as possible at the onset of training and each neuron can then eventually move into its saturation region.

3.1.1 Convolutional Neural Network Training Framework

In order to construct, train and evaluate CNNs on the gathered datasets a training framework was implemented using the C++ language. The back-end code for computing forward propagation, back-propagation and optimal learning rate computation described in Section 2.4.1 was ported to run on a CUDA graphics card in order to allow for larger and more numerous input images as well as network configurations to be used. The CUDA card used was a GTX 460 having com-

pute capability 2.x on an Intel i7-2600 machine with 12GB of RAM. This work consumed approximately 2 months of development time including debugging and testing.

The current implementation of the framework has certain limitations, requiring even numbered feature map dimensions which must also be square. This is not a particularly serious limitation, as will be show in Section 3.5.

3.2 Network Output Representation

Since we know what the target output of a network should be (a horizon or sky mask), we want to show the network input images and the target sky or horizon masks and have the network figure out the spatial filters required to generate the desired output. Since the output of the network needs to be an image, a simple approach is to let a fully connected neuron at the output layer represent a single output pixel. A drawback of this approach, however, is that for anything more than small thumbnail images, the number of connections in the network grows very quickly. If we take the example network shown in Figure 7 (Section 2.4) then we have the following contribution to the number of free parameters (weights and biases) in the network as a result of using either a simple convolutional feature map or fully connected neurons at the output layer:

	Convolutional Output Layer	**Fully Connected Output Layer**
Convolutional Layer (C1)	304	
Subsampling Layer (S1)	8	
Convolutional Layer (C2)	2020	
Subsampling Layer (S2)	40	
Output Layer (O1)	520	18,317,821
Total Network Weights	2892	18,320,193

Table 2 - Free parameter comparison between a convolutional and fully connected output layer

Therefore we see that a staggering number of free parameters are required to classify individual image pixels in the fully connected network. This removes one of the benefits of a purely convo-

lutional CNN, namely that weight sharing causes a large reduction in free parameters. In addition when porting forward and backward propagation algorithm to a GPU (which is tightly resource constrained), a large number of weights can become prohibitive; for an 8-byte double precision representation, we use over 146MB on weights alone. In addition loading, saving and managing this number of weights is prohibitive to running a large number of experiments. Finally, when observing the intermediate layers in the CNN, we notice that the network does a good job of recognizing the sky-mask even at the first convolutional layer. This is shown in Figure 25 below for a fully connected output layer network. In the figure the color channels in the HSB color space are shown first, followed by the activations of the feature maps in the first (convolutional layer). The fourth layer (convolutional) shows the network output, followed by the target output. The second and third (max-pooling) layers are omitted. We investigate the performance of both configurations (fully connected or not) and show that using a convolutional output layer is not particularly detrimental to performance.

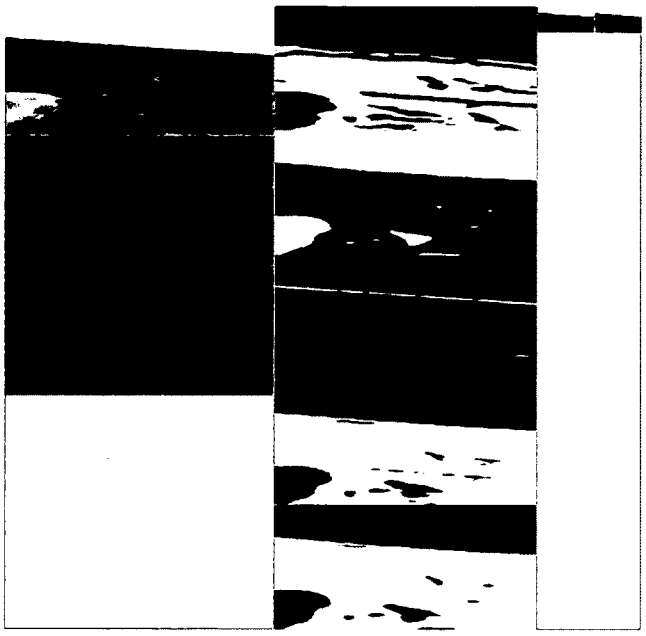

Figure 25 - Example output for a trained network to an HSB color space input pattern (leftmost column). Second column: first convolutional layer, 5 feature maps. Third column: Fully connected output layer. Fourth column: Target output

3.3 Dataset Expansion

Although several flight videos are available as training data, in the case where we are performing supervised training and attempting to generalize a trained network as best as possible, more data is always preferable. However, collecting a large amount flight video can be challenging especially in the absence of a long range test bed aircraft to carry out regular flights. Further to this, any flight videos which are available may be short or show a repeating environment; for example as in our case with a short range radio controlled aircraft. A way to generate additional video data is by applying affine transformations to existing flight videos. This is not a new approach and has been used to great effect in the literature to improve CNN training performance, for example with handwritten character recognition [54]. Although in that context, both affine and elastic transformations are common, we focus specifically on affine since the invariants keep the horizon line concurrency. In the case where a source image and its corresponding ground truth image are processed we can apply the process described in Algorithm 1.

Algorithm 1 – Image training set expansion

1. Find the center point of the original image I having dimensions WxH
2. Inscribe a circle into the original image with the same center point. The minimum image dimension M, is the diameter of the circle. So in the case where H<W we have M=H.
3. Inscribe a square, S, into the circle having a side length of floor[$M/\sqrt{2}$]
4. Rotate the original image about the viewport and sample new training images
5. Repeat steps 1-4 for corresponding ground truth images to obtain a new training pair

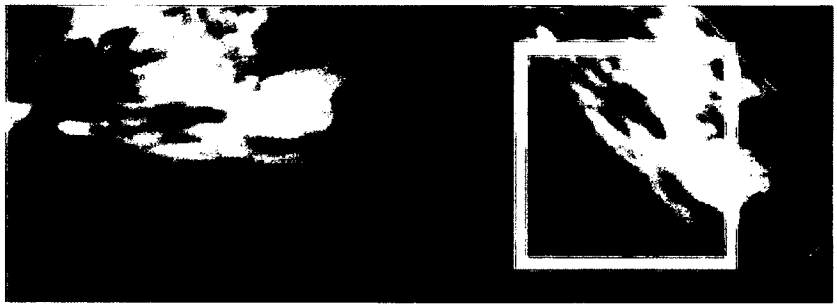

Figure 26 - Example of the original image I and rotated image with inscribed square S (highlighted in white, thickened for clarity)

The square S is the viewport for new images. By holding the viewport fixed and translating and rotating the original image I, a single image can be turned into many. An example for a rotation is shown in Figure 26. The same process needs to be applied to the target image as well, resulting in a new input/output pair. Incidentally, depending on the sizing of the input plane to a particular network we could also select a viewport of a particular size and simply sample different positions in each original image. This has the benefit of including as much information of the original image as possible since parts of the image may be unused when applying rotations.

3.4 Combining Sky and Horizon Masks

The performance of a single CNN in computing the sky-mask can be improved if we apply some post processing to the generated sky-mask, such as morphological operators [32]. However we focus specifically on a CNN only solution, combining the outputs of 2 separate networks, one

sky-mask and one horizon-mask, to produce a cleaner sky-mask. This approach is based on rewarding agreement between the sky and horizon masks.

Before combining the sky and horizon masks it can be beneficial to first post-process both images and remove any basic misclassifications. In the case of the sky-mask we expect that details such as lakes or brightly colored gravel roads may lead to some false detection of sky pixels. This may produce enclosed groups of pixels within the sky-mask which is the ground region; similarly for the sky region as well. In order to remove such erroneous regions, we can use connected components. By taking the connected components detected in the image, we can fill any child connected components contained inside a parent component with the parent components pixel value. In performing this process, we make the assumption that the validly detected ground region shouldn't be completely enclosed by a region of sky, as shown in Figure 27 below. This assumption is also made for the sky region as well since the same post-processing is also applied there.

Figure 27 - Original RGB input image, the corresponding sky-mask image, a sky-mask where the proposed approach would produce an all sky result

In order to compute the connected component tree we can use the OpenCV library. The algorithm implemented there for computing the tree is based on the work found in [100]. The basic algorithm can be summarized as follows in Algorithm 2.

Algorithm 2 – Cleanup of output image obtained from raw CNN output

1. Forward propagate an input image and obtain the output sky-mask
2. Apply the hierarchical connected component algorithm to compute the containment tree for the patches of pixels in the sky-mask
3. Flatten the tree by iterating all level 1 pixel groups in the tree and fill in the sub-groups contained by each level 1 group using the pixel intensity value of the level 1 group.
4. Generate the inverse image, with pixel values reversed (0 becomes 255 and 255 becomes 0)
5. Apply steps 2-3 but fill in the enclosed groups in the actual sky-mask instead of the inverse image

We can apply a similar approach to post-processing the horizon-mask image. The main difference is that the activation or output indication for the detected horizon with a horizon-mask image is small relative to the total area of the output image. Therefore we apply an area based method for removing horizon indications in the horizon-mask images. The area removal is based on a heuristic threshold which was determined experimentally on both the ORCC and Arnprior datasets by evaluating changes in overall classification accuracy based on changes to the minimum area size.

With both mask images post-processed, they can then be combined by looking for transitions from sky to ground in the sky-mask which agrees with regions of activation in the horizon-mask. Examples of the 2 types of outputs we're combining are shown in Figure 28 below, with the horizon-mask shown next to the corresponding sky-mask. In places where there is agreement between the two, we can process the pixels in the same column above and below the agreement region and strengthen the prediction there. Thus above the agreement point we can infer that as we move further up the image column we're more sure that subsequent pixels are sky. A similar point can be made for processing below the agreement point and enhancing ground pixels.

Figure 28 - Corresponding sky and horizon masks. The horizon-mask is 40 pixels wide with 20 pixels above and 20 below the horizon

For thinner horizon-mask images, it should be more difficult to find agreement which means a range of possible thicknesses need to be tested. In addition, multiple transitions may also be detected within the same column. In this case we can either take the transition which is closest to some number of previously made transitions, the transition which is contained by the widest contiguous activation region in the horizon-mask, or the transition having the highest vertical position. Finally both images may agree on a false location which means care must be taken in the post processing. However if we can show that the combined data approach yields consistently improved results across varying classification thresholds, on both training and test data, then we could consider these assumptions to be justified. The sky and horizon mask agreement algorithm can be summarized as shown in Algorithm 3.

Algorithm 3 - Sky and horizon mask agreement region matching process

Let M be a vector of vectors where M_i corresponds to the i-th image column and M_{ij} is the j-th valid, transition containing, region discovered in column i of the horizon-mask

for each column i in the horizon-mask image

set the current region being processed, R_{curr}, to { }. This stores the start and end points of the regions and the position where a sky to ground transition was found in the sky-mask

for each pixel p_j in the horizon mask in column i where j is the row number

if (pixel p_j in the horizon mask is a horizon region pixel)

if (not currently processing a region, R_{curr})

if (pixel q_j in the sky mask is a ground pixel)

increment j up to the next non horizon region pixel

continue

initialize R_{curr} to begin at row j having no found transitions

else

if (R_{curr} has a detected transition AND pixel q_j is a sky pixel)

increment j up to the next non horizon region pixel

set R_{curr} to { }

continue

else if (pixel q_j is a ground pixel AND R_{curr} has no transition)

set the index of transition for R_{curr} to j-1

update the end of the current region to be j

else if (currently processing a region, R_{curr})

if (R_{curr} has discovered a transition)

push R_{curr} into M_i

set R_{curr} to { }

if (currently processing a region, R_{curr})

if (R_{curr} has discovered a transition)

push R_{curr} into M_i

set R_{curr} to { }

Once Algorithm 3 is complete, M_i will be non-empty for column i if a valid transition was detected in the column. Columns with multiple transitions are then kept based on the top most transition region, although other approaches such as selecting the widest detected transition region could also be used; both of these approaches were investigated. On obtaining a unique transition point for a column, we apply a linear reward to all sky and ground pixels in the same column. If the detected transition point between sky and ground is in row i and column j and if the activation at the output layer of the sky-mask network at row i and column j is $T_{i,j}$ then we have the following update rules per column (assuming the top left of the output layer is the origin):

$$\forall i \leq I \quad T_{i,j} = T_{i,j} + \alpha(I - i + 1)$$
$$\forall i > I \quad T_{i,j} = T_{i,j} - \alpha(i - I) \qquad \text{Equation 46}$$

By adjusting the scaling parameter α we can scale the size of the update of each pixel. After completing the updates for all columns the resulting sky-mask image can be further processed by Algorithm 2 to remove any isolated regions generated by the agreement algorithm. Algorithm 3 assumes that there was valid agreement regions found. However in the case of more extreme aircraft maneuvers we may find that the horizon-mask produces no activation at all if only the ground or sky is visible. In this case we'd still like to strengthen the activations in the sky-mask based on the horizon-mask. We can do this by first checking whether the horizon-mask is indeed empty by checking if activation regions exist which are larger than some threshold β, and if there are none, we can update the sky-mask. In order to make this process more robust, we make use of the activation information in the sky-mask to inform us of how much we should weigh our

updates by taking the difference between the number of sky and ground pixels, multiplied by some scaling factor ρ. This can be written as follows with image I coordinates (i,j):

$$\forall(i,j) \in I \quad T_{i,j} = T_{i,j} + \rho(Num\ Sky\ Pixels - Num\ Gnd\ Pixels) \qquad \text{Equation 47}$$

3.5 Evaluation

The results from training CNNs of various sizes and training parameters on both datasets, ORCC and Arnprior, are given in this section. The networks tested are either 4 or 6 layer networks. This includes the input and output layers which means the tested networks have 2 and 4 hidden layers respectively; the first hidden layer is L2, the second L3, etc. In each table, where network sizing is indicated, the sizing is indicated with respect to the number of feature maps at each layer. Thus 6/6/18/18 would correspond to the first hidden layer having 6 feature maps, the second having 6, etc. Post processing is labeled as either 'basic post-processing', which refers to Algorithm 2, or as 'full post-processing' which refers to using Algorithm 2 in addition to a combination of horizon and sky masks with Algorithm 3.

As there is a level of redundancy in the datasets due to flights having been conducted within a limited flight region, it was expected that cross validation may not provide a particularly fruitful evaluation of network training performance. Instead the experiments performed in this section are based on training on each of the three datasets, using the remaining datasets as individual hold-out sets. This makes the evaluation as difficult as possible since each dataset was gathered from different locations. In particular, dataset SGL was gathered using noisy cameras during the winter while the other 2 datasets were gathered during the summer. Going forward, training accuracy refers to the accuracy obtained when training a network on one of the 3 datasets, while testing accuracy is the accuracy obtained by a trained network on a particular hold out set. Com-

puted accuracies are over the entire dataset; that is each pixel of the entire training or testing dataset is essentially classified in turn and used to populate a confusion matrix from which the accuracy is obtained.

In Table 3 the performance, in terms of accuracy, seen when varying the number of feature maps is shown. The network used for the tests has 6 layers, taking a 246x246 input image and producing a 54x54 output image; the input layer is L1, so that L2-L5 are the hidden layers followed by L6 the output layer. The learning rate used for these experiments was $0.000001(0.9)^e$ where e is the 0 based epoch index so that the learning rate steadily reduces over the training session. The number of patterns used to approximate the Hessian was 500. Each measure is given as the average of 5 runs for each network over 10 epochs, with the final network after the last epoch selected for evaluation. Finally, the table also lists the effect of applying post-processing to the network output as described in Algorithm 2.

	Dataset ORCC			Dataset Arnprior		
	Network Sizing	**Raw Acc.**	**Basic Post-proc Acc.**	**L2/L3/L4/L5**	**Raw Acc.**	**Basic Post-proc. Acc.**
YUV	4/4/12/12	96.71	96.91	4/4/12/12	94.89	95.11
	6/6/18/18	96.67	96.83	6/6/18/18	95.01	95.20
	8/8/24/24	96.9	97.01	8/8/24/24	94.32	94.45
	10/10/32/32	96.83	96.99	10/10/32/32	94.47	94.67
RGB	4/4/12/12	96.61	96.89	4/4/12/12	94.25	94.41
	6/6/18/18	96.46	96.77	6/6/18/18	93.89	94.08
	8/8/24/24	96.73	96.91	8/8/24/24	93.99	94.16
	10/10/32/32	96.37	96.54	10/10/32/32	94.55	94.73
Lab	4/4/12/12	96.28	96.55	4/4/12/12	94.99	95.32
	6/6/18/18	96.7	96.96	6/6/18/18	94.27	94.63
	8/8/24/24	96.95	97.23	8/8/24/24	93.95	94.06
	10/10/32/32	96.45	96.72	10/10/32/32	94.36	94.55

Table 3 - Effect of varying the number of feature maps at each layer

We can see that there is little variation in average performance, for both datasets, based on the input image color model. However we can see that using a post-processing always brings a small performance increase. For the false positive and negative rates, every network in Table 3 produces a higher false positive rate than false negative rate (by a factor between 1.5-2x). This can be explained from the large variation in the appearance of the ground region, for example lakes, while the sky region should have less color variation. In addition, a larger network has no effect on the training performance although the average performance between datasets does have a

consistent difference. To rule out the possibility that network size and performance are related to the selection of learning rate and the number of patterns used for Hessian computation, the same experiment was carried out using learning rates of $0.00001(0.9)^e$, $0.0001(0.9)^e$ and $0.001(0.9)^e$ and Hessian counts of 250, 1000 and the maximums for each dataset; 2427 for dataset Arnprior and 7056 for dataset ORCC.

In the case of larger initial learning rates ($0.0001(0.9)^e$ and $0.001(0.9)^e$) the networks didn't converge at all, saturating during the training process and becoming stuck in a local minima. These tests were repeated using different decay factors of 0.6, 0.8 and 0.95 showing the same results. This makes sense since we'd expect a moderately large learning rate to lend itself to a more careful exploration of the weight space as opposed to an overly large one. In the case of the number of Hessian patterns, the experiments also produced very similar results, which can be expected since the Hessian is only estimated once at the beginning of each epoch. The main difference produced by increasing the number of patterns for Hessian computation was that the time for the network to converge to a result comparable to the network produced after the final epoch; with a large number of Hessian patterns (1000 or more) this sometimes occurred during the first epoch. Example learning curves showing a plot of average MSE versus epoch index are shown for 3 different numbers of Hessian patterns in Figure 29, for 6/6/18/18 networks trained in the Lab color space on the ORCC dataset; results in the other 2 color spaces are similar. In the case of 250 or 1000 Hessian patterns, computing testing error on a network after epoch 5 and epoch 10 produced little improvement in the average over 5 trained networks (6/6/18/18); 96.42% for 250 patterns and 96.59% for 1000 patterns. When using 1000 Hessian patterns with this network configuration, 5 epochs corresponds to 1.25 hours of training time; this time includes the Hessian computation at each epoch and the loading and pre-processing of input images.

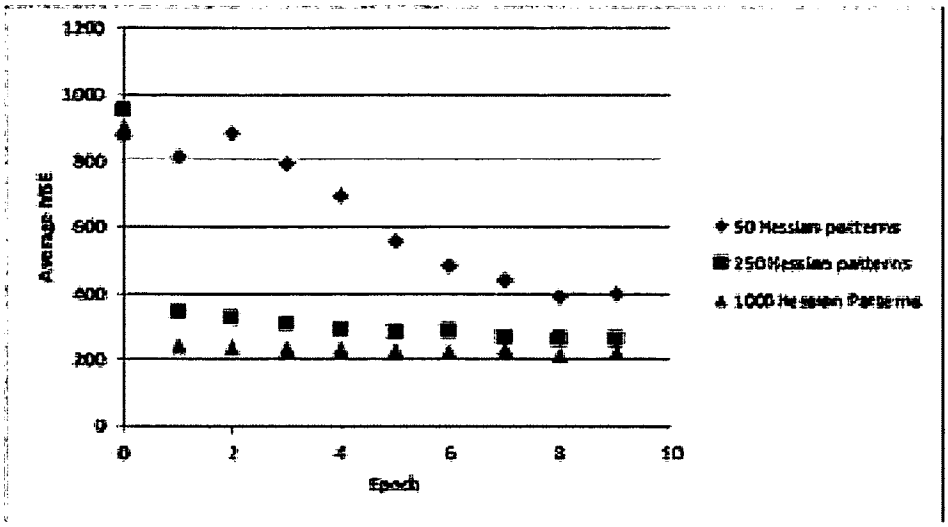

Figure 29 – Average MSE versus epoch for varying numbers of Hessian approximation patterns

In order to observe whether additional data is beneficial to training performance separate datasets were generated for each original dataset. These datasets involved generating data with either rotation, translation or both. In the case of rotations, angles from -40 to 40 degrees were used in 10 and 20 degree increments; the original dataset expands by a factor of 9 and 5 respectively. In the case of translations, the same sized viewport for generating the rotation data was used except it was shifted to the corners of the original image as well as sampling the center of it; the original dataset expands by a factor of 5. This is shown in Figure 30, with the original 246x246 sampling region in the middle and additional 4 "translation" regions at the corner of the image. The configuration used for these experiments was 246x246 Lab color space input, 54x54 output, a 6/6/18/18 configuration for the number of feature maps at each hidden layer, 500 Hessian patterns and a learning rate schedule of $0.000001(0.9)^e$; essentially the same parameter settings as used to obtain the results in Table 3.

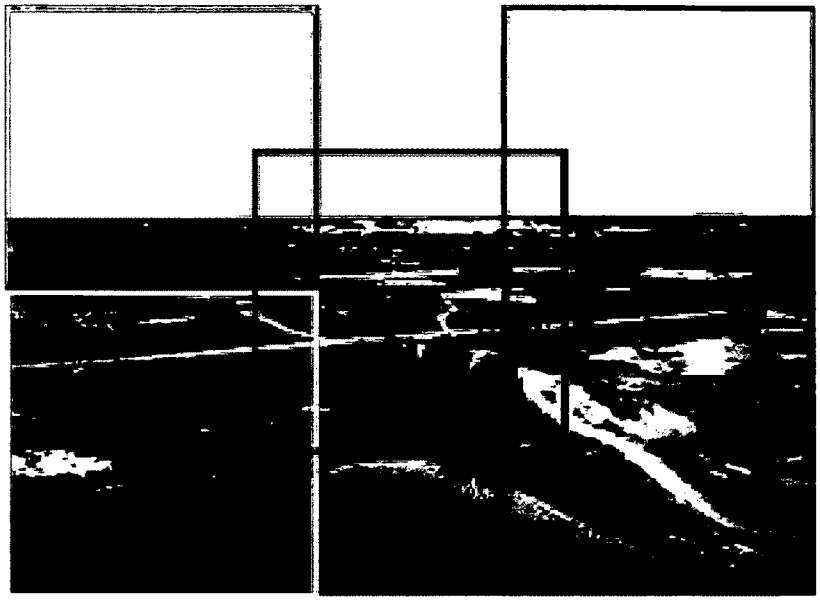

Figure 30 – Sampling regions for performing "translation" data generation

	Dataset ORCC			Dataset Arnprior		
Dataset Expansion	**Raw Acc.**	**Basic Post-proc Acc.**	**Expanded Dataset Size**	**Raw Acc.**	**Basic Post-proc Acc.**	**Expanded Dataset Size**
Shifts	97.37	97.77	35280 (x5)	95.57	95.93	35280 (x5)
Rotations	96.89	97.11	35280 (x5)	94.87	95.00	35280 (x5)
Rotations	96.71	96.83	63504 (x9)	94.81	94.97	63504 (x9)
Shifts and Rotations	97.40	97.55	63504 (x9)	95.65	95.98	63504 (x9)
Shifts and Rotations	97.70	97.99	91728 (x13)	95.33	95.87	91728 (x13)

Table 4 - Rotation and translation effects on training performance

Comparing the results in Table 3 and Table 4, data generation appears to improve the performance on both datasets, more so in the case where shifts are used. Further to this, when the net-

works which were trained on the expanded dataset were further tested on the original dataset, the same increase in performance was observed for both datasets. For example, in the case where the ORCC shift expanded dataset was tested on the original dataset, average performance over 5 trials increased to 97.51%, while for the Arnprior dataset increased to 95.44%. Although there are improvements, the fact that the increase is so small should follow in the case where the features the network is predominantly learning to extract are color instead of a more complex feature such as object edges. Indeed the training set which is used doesn't distinguish between any specific details within the 2 obstacles in question (sky and ground) but does so between them (at the horizon). In this case rotations would introduce a small number of new colors into the viewport but would be limited when compared to translations. It is more encouraging to notice that the post-processing, on average, always increases performance. Finally, the same trend of higher false positive rate than false negative rate was also observed for the resulting networks.

In order to address the issue of resolution in the output image, a shallow CNN can be created containing only 4 layers; input, convolution, pooling and output. This increases the output size by a factor of 2, reducing the number of free parameters in the process. For the results in Table 5, the parameter configuration used was the same as that in obtaining Table 3 and Table 4, including the fact that 5 separate training sessions were used to determine the averages. However while the input size remains the same, the output size is 116x116.

	Dataset ORCC			Dataset Arnprior		
Color Space	**Network Sizing**	**Raw Acc.**	**Basic Post-proc. Acc.**	**Network Sizing**	**Raw Acc.**	**Basic Post-proc. Acc.**
Lab	10/10	94.47	94.63	10/10	93.97	94.11
	15/15	95.01	95.10	15/15	94.53	94.68
	20/20	94.89	94.99	20/20	94.76	94.81

Table 5 - Training performance of a 246x246 to 116x116 4 layer CNNs on both datasets

Testing the networks in Table 5 on a shift expanded dataset (Figure 30) or training on a shift expanded dataset and testing on the original both produce similar results to those given in Table 5. For example, using 15/15 configuration networks trained on shifted Arnprior images to test on the original Arnprior dataset gives an average accuracy of 94.49%, while training on the original and testing on a shift dataset gives 94.27%. In the case of the ORCC dataset with a 15/15 configuration this becomes 94.56% for training on shifts and testing on the original, while training on the original and testing on shifts gives 94.30%. The explanation for why translations are seemingly not benefitting performance, compared to the deeper networks, is more difficult to explain however it may be due to the reduced number of free parameters for these networks. As before,

the pattern of higher false positive rate than false negative rate can also be found for these networks.

To confirm that the spatial filters being learned are predominantly based on the intensity seen by a receptive field we can flip the images in the training dataset upside down and test the performance of a trained network. This is based on the observation that if edge based filters are being learned, such that they selectively look for bright regions above darker regions, then they should be selective in their orientation. In applying both datasets to both the 6 layer and 4 layers networks, we notice that accuracy does not decrease more than 1.0% in any of the networks presented in Table 3 and Table 5.

Having observed the effects of network parameter variation and dataset sizing, it becomes important to determine the generalization ability of the trained networks. This is crucial since the UAV will eventually need to fly in new, unseen, environments. To determine this we can apply networks trained on one dataset to the images in the other, including the cases where extra data is generated; the results are shown in Table 6. From the results the out of sample error, that is the error on the other dataset, is encouraging. We observe that for a Lab color based 6/6/18/18 configuration network trained on the ORCC dataset, accuracy drops from 96.7% to 93.2% on the Arnprior dataset. In the case of a 15/15 configuration, the drop is from 95.01 to 93.65%. Similarly, in the case of an Arnprior dataset trained network, a 6/6/18/18 network drops in accuracy from 94.27% to 92.6% while a 15/15 network drops from 94.53% to 92.87%. Although it appears data generation has little effect on the out of sample error the simple post processing again, on average, generally improves results. A reversal is also noticed in the trend for a higher false positive rate than false negative rate. In particular networks trained on the ORCC dataset, while following the trend, reverse the effect when tested on the Arnprior dataset; that is, the false nega-

tive rate exceeds the false positive rate by a factor of 1.5-2x. This effect is not observed when training on the Arnprior dataset and testing on ORCC. A possible explanation for this effect is that the Arnprior dataset contains a number of images (30%) which were captured during flights late in the day and having low levels of light as shown in Figure 31. For a network trained on relatively brighter images, more detail should become to look like ground when lighting levels are decreased.

	Network Sizing	**Raw Acc.**	**Basic Post-Proc. Acc.**	**Original Dataset Size**
Trained on ORCC, Tested on Arnprior	6/6/18/18	93.2	93.41	2427 (Original)
	15/15	93.65	93.89	2427 (Original)
	6/6/18/18	92.69	92.75	12135 (x5, Shifts)
	15/15	93.84	94.04	12135 (x5, Shifts)
Trained on Arnprior, Tested on ORCC	6/6/18/18	92.6	92.75	7056 (Original)
	15/15	92.87	93.09	7056 (Original)
	6/6/18/18	92.82	92.98	35280 (x5, Shifts)
	15/15	92.61	92.71	35280 (x5, Shifts)

Table 6 - Out of sample error for trained networks on 246x246 input images and 54x54 (4 hidden layer) and 116x116 (2 hidden layer) output images

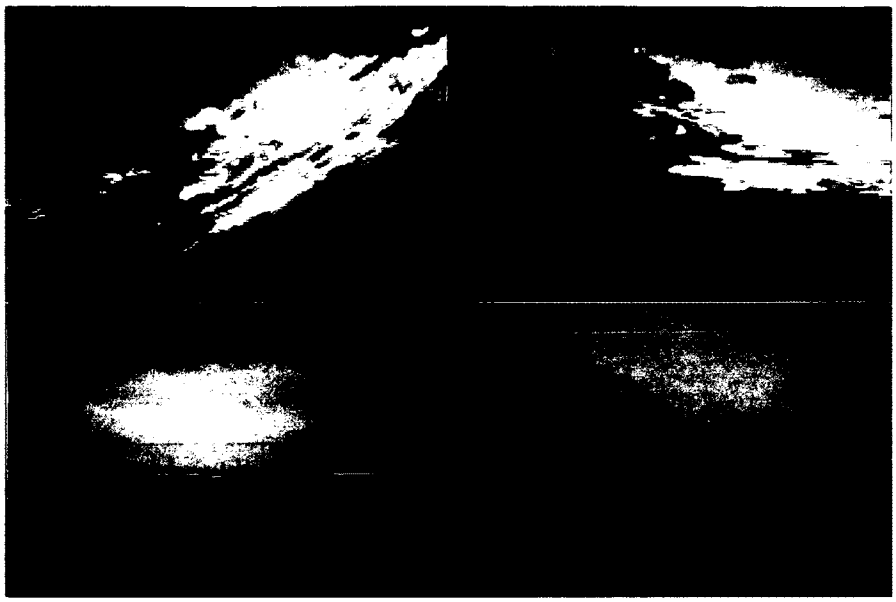

Figure 31 – Example images containing low levels of light in the Arnprior dataset

When networks with subsampling layers were trained for sky-mask generation, the training process did not converge. The reverse effect was observed for training networks with max-pooling layers on horizon-mask datasets. This suggests that throwing information away (in the case of max-pooling) is beneficial to producing activations across an entire image while not doing so (subsampling) works only for specific image regions. In the case of training horizon-mask networks, we use the same learning rate, Hessian pattern count and training stopping time as in previous examples; as mentioned subsampling layers were used instead of max-pooling as well. The results of testing on the opposite dataset are summarized in Table 7 where 2 different network depths (4 and 6 layers) were used. The results can be seen to be comparable to the results in Table 6 for the out of sample performance of sky-mask networks. It is important to note that the evaluation metric does not take into account the cases where there the image is all sky or all ground. In this case a qualitative evaluation was instead performed, where it was noticed that a spurious horizon line was never produced in such images across both datasets.

Network Sizing	Acc. when trained on ORCC, tested on Arnprior	Acc. when trained on Arn-prior, tested on ORCC
5/5/15/15	94.3	93.85
8/8/24/24	94.6	93.69
5/5	93.2	92.97
8/8	93.54	93.07

Table 7 - Out of sample test performance for horizon-mask networks

Finally, the performance gained from combining horizon and sky mask networks is shown in Table 8. The networks used are those for which the performance was given in the previous tables in this section. Values of $\alpha = 0.0375$ (Equation 46), $\rho = 0.25$ (Equation 47) and $\beta = 50$ (the maximum area of pixel blocks which are considered to be noise in horizon mask images) were used to obtain these results and were obtained through trial and error. Although the overall increase in performance with post processing seems small compared to the raw result, we see that when testing on the opposite dataset the post processed results are comparable to the training performance on the opposite dataset. The increase comes with virtually no computational cost, requiring less than 6ms to run, even with 116x116 output images.

		Trained on ORCC		Trained on Arnprior	
	Sky and horizon network sizing	**Raw Acc.**	**Full Post-proc. Acc.**	**Raw Acc.**	**Full Post-proc. Acc.**
Tested on ORCC	6/6/18/18 and 5/5/15/15	96.7	97.41	92.6	93.27
	15/15 and 5/5	95.01	95.67	92.87	93.86
Tested on Arnprior	6/6/18/18 and 5/5/15/15	93.2	93.97	94.27	94.98
	15/15 and 5/5	93.65	94.59	94.53	95.12

Table 8 - Post processed sky-mask performance when combining sky and horizon mask networks

Dataset SGL was also evaluated using 246x246 input and 116x116 output images. When testing the networks presented in Table 8 we obtain 91.48% accuracy when testing this dataset on a 15/15 network trained on the ORCC dataset and 90.55% on an Arnprior dataset trained 15/15 network (these measures include basic post-processing). The basic post-processed training performance of an SGL dataset 15/15 network is 93.11% for comparison. Applying this network on either the ORCC or Arnprior dataset yields very poor results however, giving 43.17% and 44.03% respectively. It is difficult to ascertain whether a lack of data or a lack of color in the images is the underlying problem (generating additional data using shifts only decreased performance), although the latter is believed to be the problem. On the other hand the performance of ORCC and Arnprior dataset networks is very promising considering that more than half of the images in the SGL dataset have artifacts introduced due to the interlaced cameras used to capture

the sequence. Thus it only confirms the ability of these networks for the horizon and sky mask detection task.

3.5.1 Performance Comparison with SVMs

To apply SVMs for generating a sky-mask, the input and target images in both datasets were first subsampled using 246x246 viewports. The windows were then down-sampled to 54x54 images which correspond to the output sizing of a number of tested networks in Section 3.5. Each pixel in the 54x54 image represents a feature vector, as explained in Section 2.3.2, and is computed over a window around each pixel; window sizes of 3x3 and 7x7 were tested. For a 3x3 window, a 52x52 sky-mask would then be produced, neglecting the boundary pixels, with 52^2=2704 feature vectors per image. In the case of the ORCC dataset, 7055 images meant over 19 million training examples. This is potentially good news in terms of the out of sample error estimate (assuming a relatively small number of resulting support vectors) however in terms of training time and computational resources it meant that attempting to train on image sizes beyond 54x54 was not feasible; specialized, GPU oriented, SVM training algorithms were not investigated.

Two types of SVM kernels were used for all tests; linear and radial basis function (RBF). In the case of the RBF kernel, the scaling factor used for the exponential was 1. The soft-margin cost value C used for experiments was the MATLAB default of $\frac{N}{2*N_1}$ for class 1 (sky) and $\frac{N}{2*N_2}$ for class 2 (ground) where $N = N_1 + N_2$. The training algorithm used was SMO since the QP algorithm was unable to execute on either dataset due to lack of memory (12GB); the maximum number of SMO iterations was 1 million. All inputs were shifted and scaled to make their mean 0 and standard deviation 1 before training.

When training on the ORCC dataset an accuracy of 97.63% for the RBF kernel using 3x3 windows was observed while with a 7x7 window this increased to 97.98%. This contrasts sharply with the linear kernel SVM which produced 39.97% and 43.66% for the 3x3 and 7x7 windows respectively, indicating the difficulty of data separation by the linear kernel. In the case of the smaller Arnprior dataset, 96.83% and 97.53% was observed for the 3x3 and 7x7 windows respectively for the RBF kernel based SVM. In the case of the linear kernel the accuracy fell to 22.9% and 23.47% with 3x3 and 7x7 windows respectively.

When testing trained (RBF kernel) networks on the other dataset we find, for the ORCC dataset trained networks tested on the Arnprior dataset, network performance dropping to 94.23% and 94.41% for 3x3 and 7x7 windows respectively. In the case of Arnprior dataset trained networks tested on the ORCC dataset, the accuracy becomes 93.23% and 93.87% for the 3x3 and 7x7 windows respectively.

When testing on the SGL dataset using 3x3 window RBF kernel SVMs trained on the ORCC and Arnprior datasets, we see out of sample error rates of 73.21% and 68.49% respectively. This contrasts sharply with the 91.5% and 90.5% rates obtained by applying a CNN trained on the respective datasets.

We can see that for both ORCC and Arnprior datasets, CNNs are comparable in performance, within 1% accuracy, to the RBF kernel based SVMs. The number of generated support vectors is quite large however, for example producing over 500,000 for the Arnprior dataset. This may be a problem when considering a hardware based SVM implementation which is severely resource constrained. In terms of runtime performance it is important to remember that the evaluation performed with SVMs was purely CPU based (MATLAB) while the evaluation for CNNs was on

GPU hardware. With that in mind, we see that the training time for both linear and RBF kernel SVMs on the Arnprior dataset was considerable, requiring 4 and 6 days respectively to train. The time to classify for the linear SVM was reasonably fast taking approximately 0.5 seconds per input pattern while taking over 2 seconds for the RBF kernel. Training time could potentially be an issue in cases where rapid site surveying is performed, requiring frequent flights to be performed which produce new training data.

Although the performance results appear bleak, research can be found in the literature on porting both SVM training and evaluation to a GPU [75]. The results from this work show speed-up's ranging between 9-35x on CUDA based GPU's, although the performance is usually not evaluated on datasets bigger than a few hundred thousand input vectors.

3.6 Platform and Runtime Performance Considerations

CUDA based graphics hardware was used in this thesis to train and test CNNs. This was driven by the knowledge that one of the future GeoSurv II capabilities will be to provide onboard power to the obstacle detection system.

In the literature GPU based applications of CNNs have typically focused on relatively small input images sizes [68] or older CUDA hardware (compute capability 1.x) [69], [70], [71]. Investigating the performance of implementations on older CUDA hardware is still informative for 2 reasons. The first is that the reported performance can be seen to be quite good under the appropriate optimization [72], being able match or even outperform our un-optimized implementation. For example in [72] Strigl et al. report a forward propagation performance of approximately 1.5ms for 104x104 grayscale images for a LeNet-5 style network while Scherer et al. report a runtime of approximately 10ms for 256x256 grayscale images with a similar network configura-

tion. A corresponding performance curve for a convolution only network and also the LeNet-5 variant using varying size input images is given in Table 9 for our implementation. The second reason is that mini and micro ATX based systems with integrated CUDA cards are available on the market, for older (1.x) and some newer (2.x) compute capabilities. These systems still draw a large amount of power (150-200W) however the newer NVIDIA ION based systems can draw as little as 30W [94]. Since these systems support CPU's, hard drives, Ethernet and USB ports they offer an attractive all in one platform for supporting both the obstacle detection and avoidance algorithms as well as any peripherals such as cameras or interface to the MicroPilot in the GeoSurv. Nonetheless, it is informative to investigate other potential platforms for deploying CNNs. When comparing CPU and GPU based CNN implementations, the GPU based implementations typically outperform their CPU counterparts by a 10-20x speed-up [69][72].

Network Sizing (L2/L3/L4/L5) and (Input size to Output size)	Run-time Performance (ms)
8/8/24/24 (146 to 29)	4.49
8/8 (146 to 66)	2.28
8/8/24/24 (246 to 54)	12.89
8/8 (246 to 116)	6.48

Table 9 - Performance Curve for Varying Input Sizes

In addition to GPU's, CNNs have also been ported to a low-end DSP-oriented Field Programmable Gate Arrays (FPGAs) [65]. In [65] Farabet et al. describe a programmable CNN Processor with an instruction set which matches the elementary operations of a CNN. The system is shown to produce a performance of 100ms per forward propagation on 512x384 grayscale images and a 7 layer network using the Spartan-3A DSP 3400 and Virtex-4 SX35 FPGAs. The work in [65]

improves on the work by Cloutier et al. in [66] which was restricted to limited capacity FPGAs with low accuracy arithmetic. A further analysis of FPGA CNNs for large scale CNNs with 500x500 input images is further given by Farabet et al. in [71]. In [67] Chakradhar et al. present an FPGA based implementation on a Virtex 5 which is able to process 640x480 grayscale images at speeds in the range of 33 to 40ms using network sizing similar to that of [65]. A significant advantage of deploying a CNN on an FPGA is the low power consumption compared to GPU's, requiring as little as 15W [65], [67], [71].

3.7 Conclusions

In this chapter a new approach to horizon and sky mask generation using convolutional neural networks was presented. The results show accurate classification of sky and ground pixels across 2 large real-world datasets containing various terrain as well as on a third, small, dataset which was taken with a different camera, seasonal conditions and in the presence of significant noise. No particular color model was found to out-perform any other consistently across the network configurations which were tested. As such particular color model is endorsed, the selection of which should depend on the application. For example if visual evaluation of input images is critical, the RGB color space would be appropriate.

4 Edge Feature Based Stereoscopic Obstacle Detection

This chapter expands on the edge feature based stereo obstacle detection pipeline introduced in Section 2.2.2. This includes combining the edge based feature images with the sky-mask generated by a convolutional network as outlined in the previous chapter, followed by stereo corre-

spondence and 3D mapping of detected obstacles. A small number of stereoscopic videos are used to evaluate the proposed approach.

4.1 Experimental Setup

Two pairs of cameras were used for collecting all datasets used in this thesis. The first pair was shown in Figure 3 in Section 2.1.2, and records 640x480 video at a frame rate of 30 frames/second. These cameras have a focal length of 3.6mm and horizontal and vertical angles of view of approximately 50 and 40 degrees respectively. The second pair of cameras are Logitech C250 webcams. They have a resolution of 640x480 with a CMOS sensor and a focal length of 2mm. They record at 30 frames/sec and have horizontal and vertical angles of view of approximately 50 and 40 degrees respectively. The first pair of cameras was used during a Telemaster flight test at the Arnprior RC field to track a group of balloons. Since the data collected during this scenario was short, a number of ground tests using webcams and a laptop were performed with balloons and other artificial structures above the horizon acting as potential obstacles. Calibrations were performed using a reference checkerboard pattern for all obtained sequences using the MATLAB Camera Calibration Toolbox.

4.2 Obstacle Datasets

Three stereoscopic video sequences were obtained for testing the algorithms in this section. The first was a clip from a test flight at the Arnprior airfield, showing a low approach towards balloon obstacles. An example stereo image pair from this sequence is shown in Figure 32 with the locations of the balloon obstacles circled for clarity. An overhead view of the Arnprior airfield is shown in Figure 33 noting the starting position of the aircraft at the beginning of the video (Y_1), the final position (Y_2) and the location of the balloon obstacle (O_B). It is important to note that the use of overhead imagery as a baseline for distances as well as the marking of specific map

points to correspond to the locations of captured video is only meant to provide a rough, order of magnitude, validation of the depth estimates generated by the proposed method.

Figure 32 – Stereo image pair from Arnprior video sequence (balloon obstacles location circled in red for clarity)

Figure 33 – Overhead image of the Arnprior airfield with approximate initial aircraft location (Y_1, red), end location (Y_2, red) and balloon obstacle location (O_B, blue). Distance scale shown bottom left.

An overhead view of the location for the ground tests is shown in Figure 34. The location of the balloon obstacles is marked as O_B (in blue) in the image. The ground tests consisted of 5 sequences, beginning at various locations on the field which are marked as X_1 to X_5 in Figure 34, walking towards the various obstacles with a stereo camera pair; the walk duration is short (all videos are less than 1 min. in length) and in the directions indicated by the arrows. The obstacle

locations in the various videos are marked as O_1-O_6 as well as O_B for the balloon obstacles. For example Figure 35 shows O_B and O_6 for the walk starting at X_1, while Figure 36 shows obstacles O_1 to O_4 for walk X_4. Obstacle O_5 in Figure 36 is occluded due to obstacle O_1.

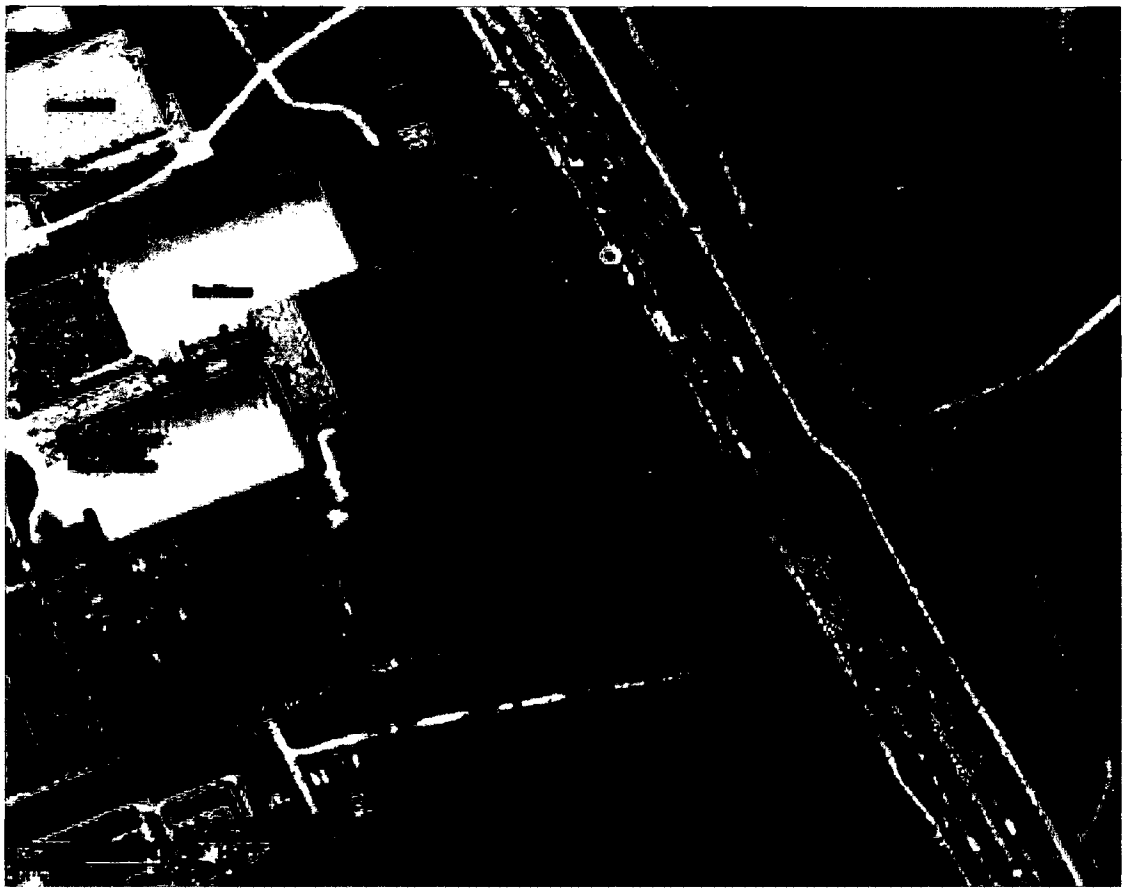

Figure 34 – Aerial view of ground test location (distance scale, bottom left corner)

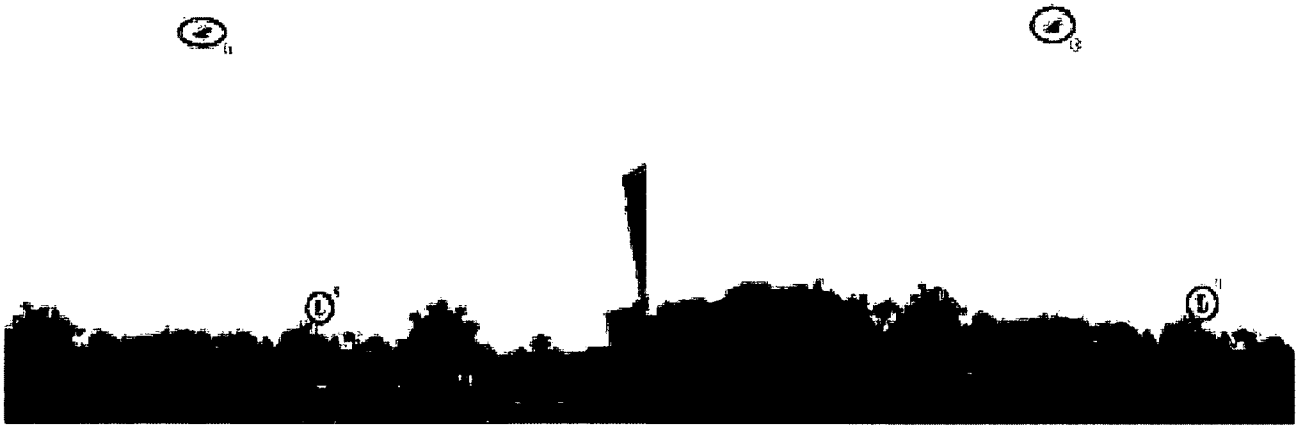

Figure 35 – Stereo image pair from a ground test sequence with above horizon obstacles circled and numbered

Figure 36 – Another stereo image pair from a ground test sequence with corresponding above horizon obstacles and occlusion circled and numbered

4.3 Feature Extraction

This section outlines the feature extraction stage in the obstacle detection pipeline shown previously in Figure 4. The Sobel edge filter was applied to all captured sequences presented in Section 4.2. Frames in each sequence were manually checked for detection of objects of interest, such as those circled in Figure 32, Figure 35 and Figure 36, over the length of each sequence. This was performed over a number of cutoff thresholds for the edge gradient magnitude to investigate the detection and noise rates for each sequence. The validity of assuming that the edge maps produced for a particular obstacle between a stereo image pair is deferred to Section 4.4. The detection rate of objects, manually identified to be potential obstacles, is summarized in Table 10 below. The corresponding starting point for each sequence and the objects visible throughout the sequence in both views are given. The combined detection rate gives the percentage of stereo frames in which each sequence object was both visible and detected in both the left and right stereo frames. Visibility was manually determined for each frame, which was possible since all sequences were 1 minute or less in duration.

Sequence	Starting Point	Sequence Objects	Combined Detection Rate
1 (Flight)	Y_1	O_B	98%
1 (Ground)	X_1	O_B, O_6	100%
2 (Ground)	X_2	$O_1 - O_5$	100%
3 (Ground)	X_3	O_B	100%
4 (Ground)	X_4	$O_1 - O_5$	100%
5 (Ground)	X_5	O_B, O_6	100%

Table 10 – Combined object detection rates for all video sequences (gradient threshold of 64)

Decreasing the gradient threshold to 32 allows for 100% detection rates for all objects; however a larger amount of noise appears in the resulting edge maps. Examples of using a threshold of 64 are shown below in Figure 37 for the flight sequence. A typical example image with cloud cover, taken from the ORCC database is shown in Figure 38. The threshold used in this image is the same as that used to obtain the results in Table 10. Although further flight tests videos are necessary to better understand the effects of cloud cover and threshold, it appears that under favorable weather conditions a threshold similar to that used in Figure 37 and Figure 38 would be appropriate to both detect small, high contrast, obstacles and also be robust to noise from cloud cover.

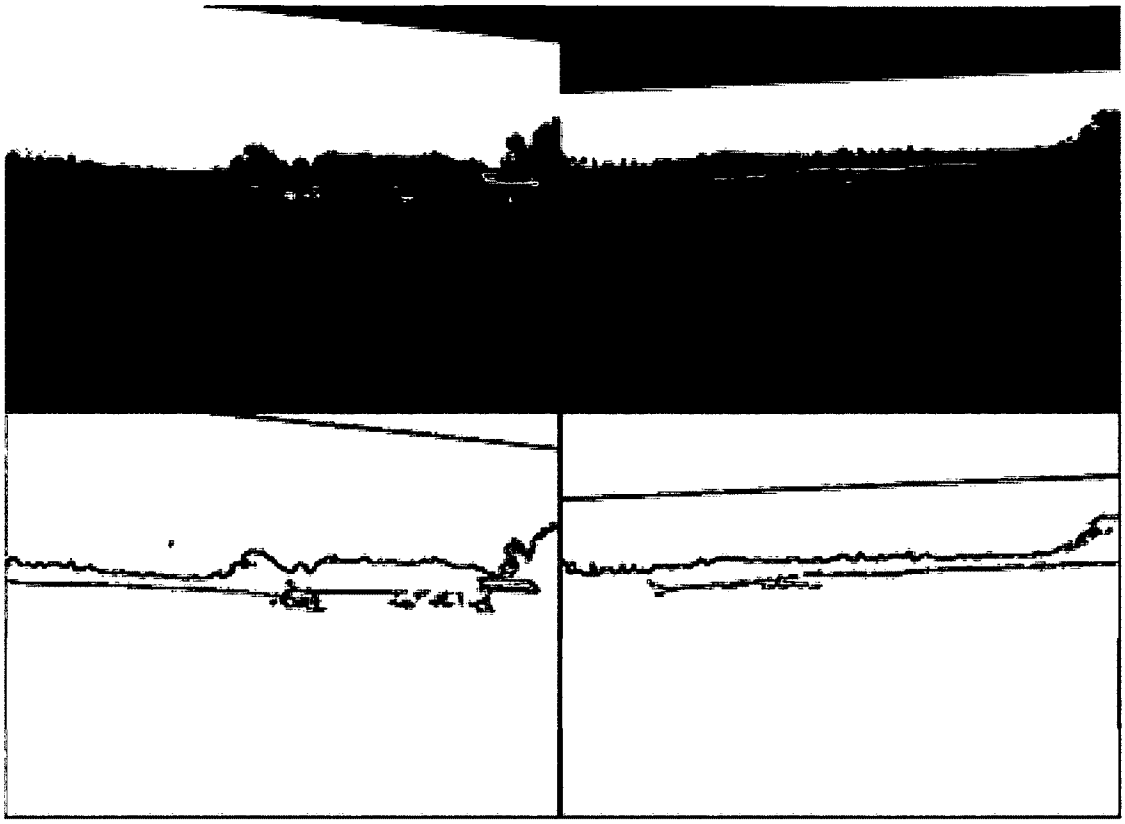

Figure 37 – Original stereo image pair (top) and the corresponding Sobel edge maps (bottom) with the balloon distinguishable above the horizon (gradient threshold of 64). Edge maps inverted for clarity.

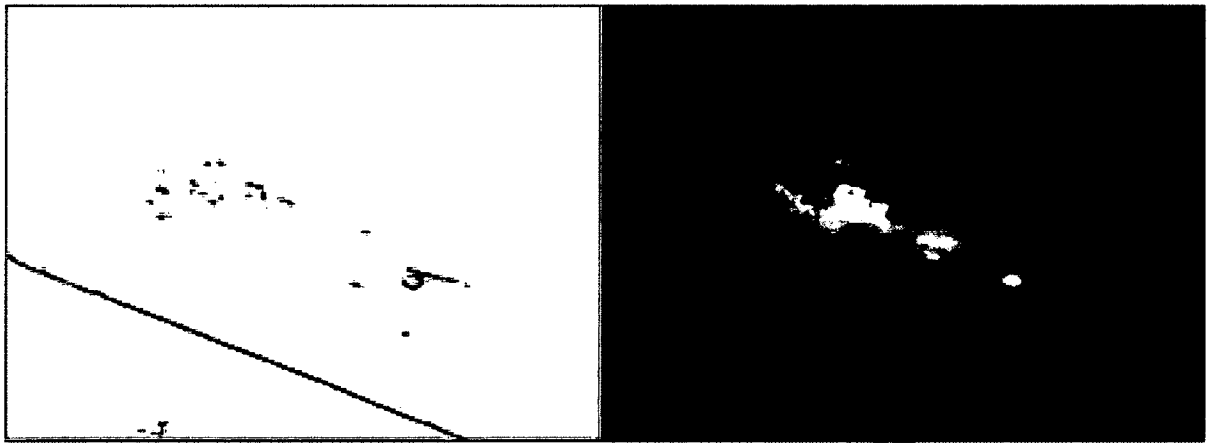

Figure 38 – Example image with could cover from the ORCC database (gradient threshold of 64). Edge map inverted and highlighted in red for clarity.

4.4 Sparse Block Matching

This section focuses on presenting the block matching results on extracted edge features. Two sequences are focused on in particular. The first is ground sequence #4 due to the abundance of visible obstacles as well as the flight sequence since it is the only calibrated stereo flight sequence with above the horizon obstacles which was successfully captured. Shown below in Figure 40, Figure 41 and Figure 42 are the example, consistency checked, left disparity maps for a rectified stereo image pair (Figure 39) from ground test sequence #4 using each measure discussed in Section 2.5.4.2. In the case of ZNCC and SAD, a window size of 5x5 was used for the correspondence. The image pair was the first captured in the sequence, corresponding to location X_4 in Figure 34. The sparseness in disparity can be seen for both the BT and ZNCC measures, while the SAD measure does not seem to suffer from this problem allowing it to detect obstacle O_2 and O_3 as well (circled in red in Figure 42 and originally shown in Figure 36). Looking closer at the computed disparities for obstacle O_1 in Figure 42, the disparity observed over the pixels of the obstacle is 11. Using the calibration produced re-projection matrix Q of:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -326.627 \\ 0 & 1 & 0 & -232.671 \\ 0 & 0 & 0 & 759.056 \\ 0 & 0 & 1.51875 & 0 \end{bmatrix}$$

A disparity of 11 produces a depth of 45m (with a range resolution of 4m). Consulting Figure 34, we see that the ground truth distance for the starting point of the sequence is approximately 37m. Looking at obstacle O_2, we notice a disparity of 4 which corresponds to a depth of 125m (with a range resolution of 31m). From Figure 34 we expect a depth of approximately 100m. Obstacle O_3 was also detected, although disparities of both 1 and 2 were observed for the obstacle; this corresponds to depths of 500m and 250m respectively. The corresponding depth resolutions for

these depths are 500m and 125m respectively. The true depth from Figure 34 is approximately 250m. The trees to the right of obstacle O_1 vary between 150m and 200m in Figure 34, with their representation being sparse due to the large and cluttered variation in their shape. Their corresponding disparity varies between 3 and 5 or 166m and 100m (corresponding resolutions of 54m and 22m).

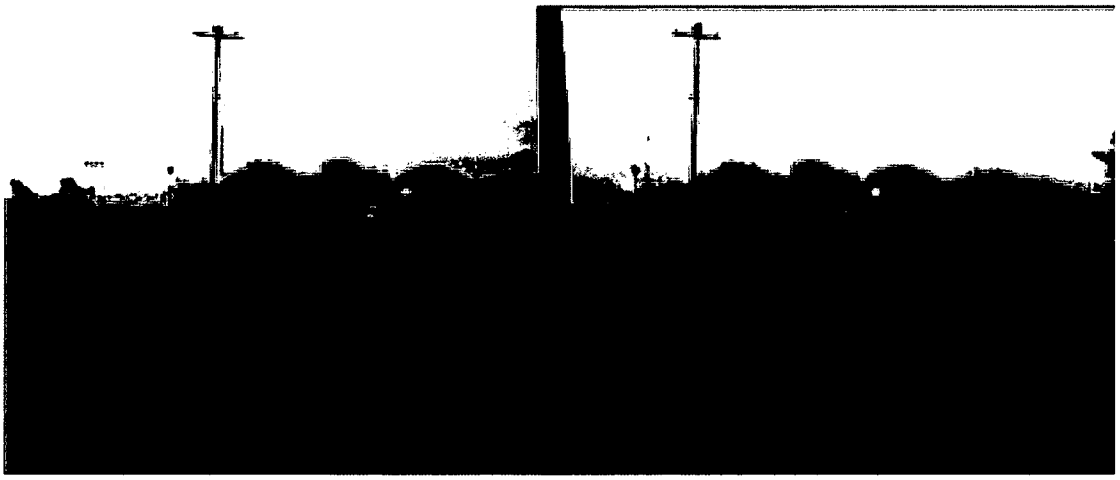

Figure 39 – Rectified stereo image pair from ground sequence #4

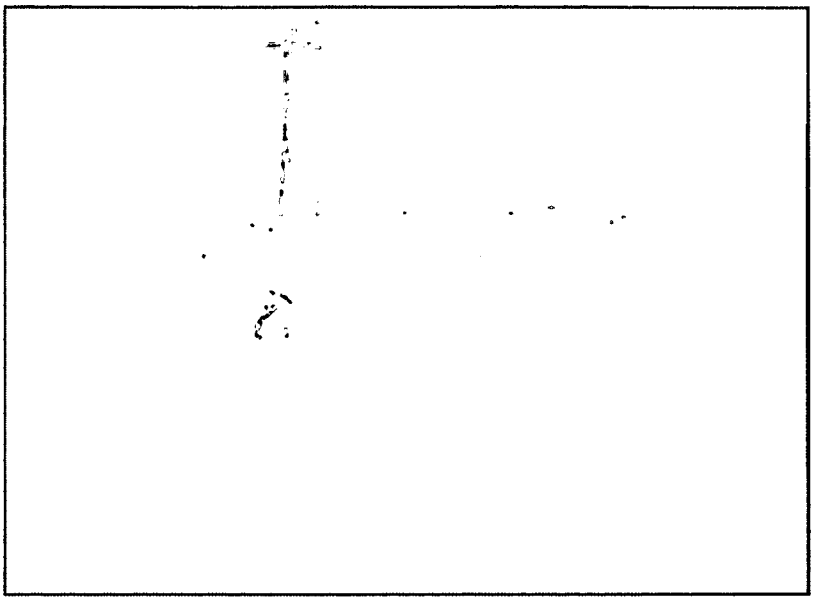

Figure 40 – Birchfield and Tomassi (BT) measure based disparity map (darker color is closer to the cameras as the image is inverted, disparity search range of 1-20 and edge gradient threshold of 64).

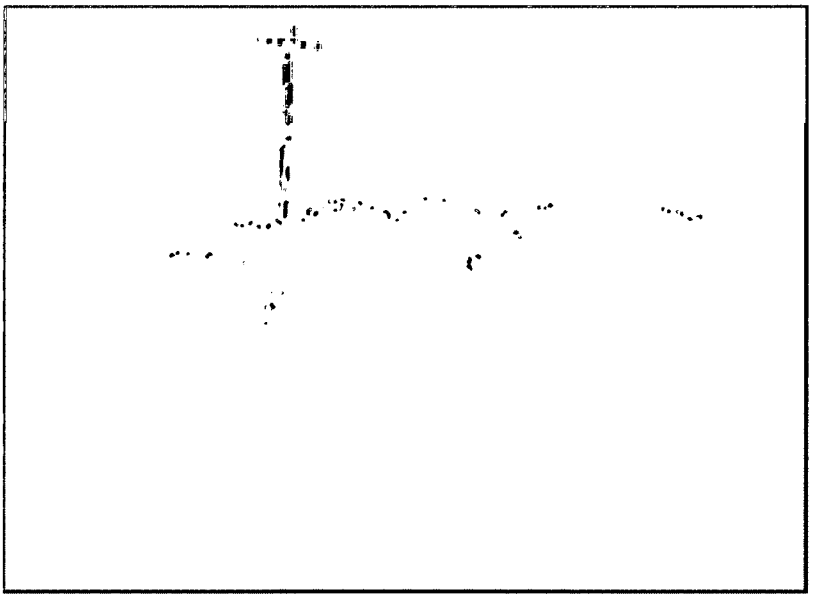

Figure 41 – ZNCC measure based disparity map (darker color is closer to the cameras as the image is inverted, disparity search range of 1-20 and edge gradient threshold of 64)

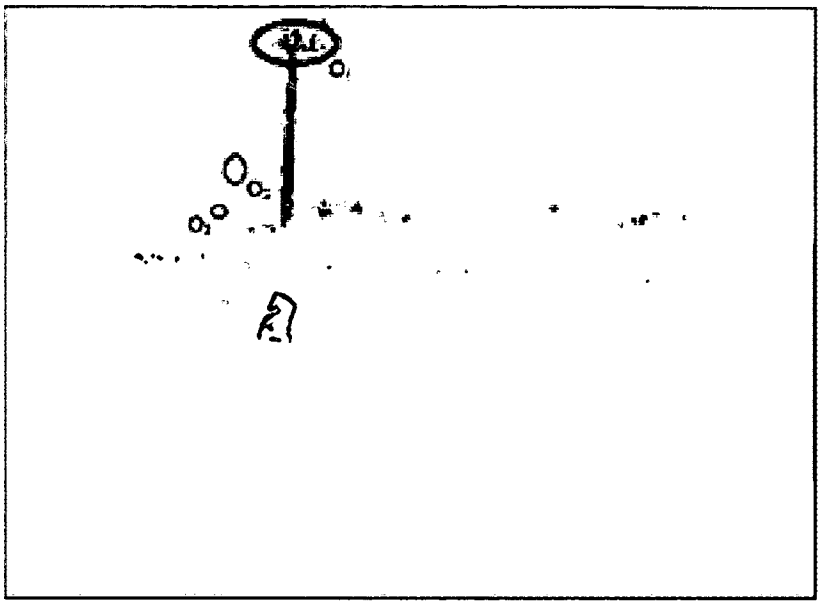

Figure 42 – SAD measure based disparity map (obstacles O_1, O_2 and O_3 from Figure 36 circled in red, darker color is closer to the cameras as the image is inverted, disparity search range of 1-20 and edge gradient threshold of 64)

The BT and ZNCC measures produce similar results in the other ground sequences and fail to agree on the balloon obstacle disparity in the flight sequence, missing the balloons entirely; similar results are observed using window sizes of 3x3, 7x7 and 9x9. Thus it appears for both camera pairs the issue of gain offsets and bias is not a pressing one to correct. It was generally observed that in the case of the SAD measure, disparity results became more uniform in spatially similar regions when using larger search windows (9x9 and up). However small details, such as obstacles O_2 and O_3 in Figure 42 which are only a few pixels in size, disappear entirely after the consistency check with window sizes larger than 5x5 which is to be expected since surrounding detail becomes more dominant and eliminates small features.

In applying the SAD measure with a 5x5 window to the flight sequence (example rectified stereo pair shown in Figure 43) the disparity map shown in Figure 44 is obtained. Although the disparity map is quite sparse, the balloon obstacles are detected as well as a small number of horizon

features. The visible correspondence matches at the horizon correspond to disparities of 1 and 2 and, using the re-projection matrix given below, correspond to respective depths of 1200m and 600m (respective uncertainties of 1200m and 300m). Comparing the aircraft position between Figure 33 and Figure 43, the location of the farthest trees in the direction of the flight path are approximately 650m away. In the case of the closest trees to the right of the flight path, we can measure they are approximately 250m away, while the observed disparities are 4 and 5, corresponding to depth of 300m and 240m respectively (with respective resolution errors of 74m and 47m). Finally, the balloon obstacles are approximately 200m away in Figure 33 while their measured disparity is 6, corresponding to a depth of 200m (range uncertainty of 33m). While the sparseness of disparity results comes from the strict consistency check being applied, the correspondence between the estimated disparities and those estimated from overhead views is encouraging.

$$Q = \begin{bmatrix} 1 & 0 & 0 & -228.034 \\ 0 & 1 & 0 & -151.307 \\ 0 & 0 & 0 & 931.861 \\ 0 & 0 & 0.77377 & 0 \end{bmatrix}$$

Figure 43 – Rectified stereo image pair for the flight sequence

Figure 44 – Disparity map computed using SAD 5x5 block matching with annotated obstacles circled with respect to Figure 43 (disparity range 1-16, gradient threshold 64)

To look closer at consistency of re-projected points in 3D with physical measurements we can take the left view from the rectified views in Figure 39 and mark the pixel locations of specific features on the closest and largest obstacle (the lighting pole) as shown in Figure 45. The pixel locations of each point is seen to be A = (201, 26), B = (227, 30), C = (251, 33), D = (216, 232) and E = (215, 270). The physical distance between points D and E was manually measured to be

1.9 meters and using this measure the physical distance between points D and B was estimated to be 11.4 meters. Using the re-projection matrix and the disparity of 11 found for the obstacle, the corresponding 3D locations (indicated by adding a prime symbol to each point) of the indicated pixels can be computed as being A' = (-7.52, -12.37, 45.43), B' = (-5.96, -12.13, 45.43), C' = (-4.52, -11.95, 45.43), D' = (-6.62, -0.04, 45.43) and E' = (-6.68, 2.23, 45.43) where each coordinate is in meters. These results allow us to compute that the vertical distance between D' and E' as 2.27 meters while the distance between D' and B' is 14.36 meters. These estimates are a reasonable approximation of the true, measured, height of the physical object.

Figure 45 – Feature points used to estimate the validity of the re-projection

It is important to consider the detection rate of manually identified obstacles by the stereo correspondence algorithm as it gives an idea of the percentage of the total number of image frame where at least a single disparity was computed for a particular obstacle. These rates can be compared to those given in Table 10, which give the detection rates for manually identified obstacles in terms of edge detection. Since the consistency check is strict and is aimed at removing noise,

we expect that the stereo processed detection rates are lower than the feature extracted rates. This is confirmed by measurements and is summarized for all sequences in Table 11. Rates are presented as percentages relative to the total number of frames during which the particular obstacle is visible in both the left and right views as well as having been detected by the feature extraction processing stage as well. From the measurements, we notice that the detection rate for the balloon obstacles drops from 98% to 93%, where the drop is in cases when rapid illumination changes occur. In the case of ground sequences 2 and 4, we notice that the farthest obstacle (O_3) drops in detection rate from 100%. This is primarily due to its small size due to its large distance from the cameras (250m with a range resolution error of 125m).

Sequence	Starting Point	Sequence Objects	Combined Detection Rate
1 (Flight)	Y_1	O_B	93%
1 (Ground)	X_1	O_B, O_6	100%
2 (Ground)	X_2	$O_1 - O_5$	100% (O_1, O_2, O_4, O_5), 94% (O_3)
3 (Ground)	X_3	O_B, O_6	100%
4 (Ground)	X_4	$O_1 - O_5$	100% (O_1, O_2, O_4, O_5), 91% (O_3)
5 (Ground)	X_5	O_B	100%

Table 11 – Stereo detection rates (as percentage of total frames) for all manually identified obstacles (gradient threshold of 64, SAD window size of 5x5, disparity range 1-32)

4.5 Horizon Boundary Application and Model Construction

As stated in Section 2.2.1, the goal of this thesis is to detect obstacles both at and above the horizon. Having obtained an accurate sky-mask using the process outlined in Chapter 3, it should be possible to achieve the latter by selecting the sky region of the image for further feature extraction and stereo correspondence. However if we consider the cases shown in Figure 46 below, where obstacles form part of the horizon, we can see that an accurate sky-mask will actually hin-

der our detection efforts for obstacles at the horizon since our sky-mask will not retain features exactly at the horizon.

Figure 46 - Example collision scenarios with the collision obstacle forming part of the horizon boundary (left image: building and tree tops, right image: tree tops and poles)

In order to ensure that features just below the horizon boundary do propagate to the stereo correspondence stage, we use the data computed during the agreement processing stage between sky and horizon masks (described in Section 3.4). There we tabulated all agreement regions where a valid sky to ground transition was detected in the sky-mask which agreed with the horizon-mask. By appending the horizon-mask to all columns of the sky-mask in which there is agreement between the two, both the horizon boundary and the immediate region below it can be combined together. Square regions from the original input images were taken in order to overcome the restriction that the inputs to the CNN framework must be square. For 640x480 images this produced 480x480 sub-regions in the original images. The 480x480 images were then downsampled to 246x246 and applied to 4 layer networks described in Section 3.5, producing 116x116 outputs. The 116x116 outputs (for both sky and horizon masks) were then logically

added together with the resulting image up-sampled to 480x480. An example RGB image with sky and horizon masks generated by a 4 layer CNN trained on the Arnprior dataset, up-sampled from 116x116 to 480x480, is shown in Figure 47. Note the results produced appear quite good even though the network has never seen the given image which was also taken using a different camera than that used to capture the ORCC and Arnprior datasets. The corresponding edge map is shown in Figure 48 along with the results of masking with the sky-mask and a combination of sky and horizon masks together shown as well. The combination mask allows the entire edge mask to appear.

Figure 47 - Example of an original image (left), a sky-mask (center) and horizon-mask (right) image generated by a CNN and up-sampled

Figure 48 - Example of an original edge map (left, gradient threshold 32), masked with the sky-mask in Figure 47 (center) and masked using a combination of sky and horizon masks (right). Images are inverted and highlighted in red for clarity.

Another example similar to the one above is given for an image from the flight sequence in Figure 49 and Figure 50. From looking at the sky-masked edge map it is clear where the false positives are, as these pixels show up as horizon edges from the original edge map. Adding in the horizon-mask allows for the clutter below the horizon to be removed, albeit still passing through a small amount of detail below the horizon.

Figure 49 – Example of an original image (left), a sky-mask (center) and horizon-mask (right) image generated by a CNN and up-sampled

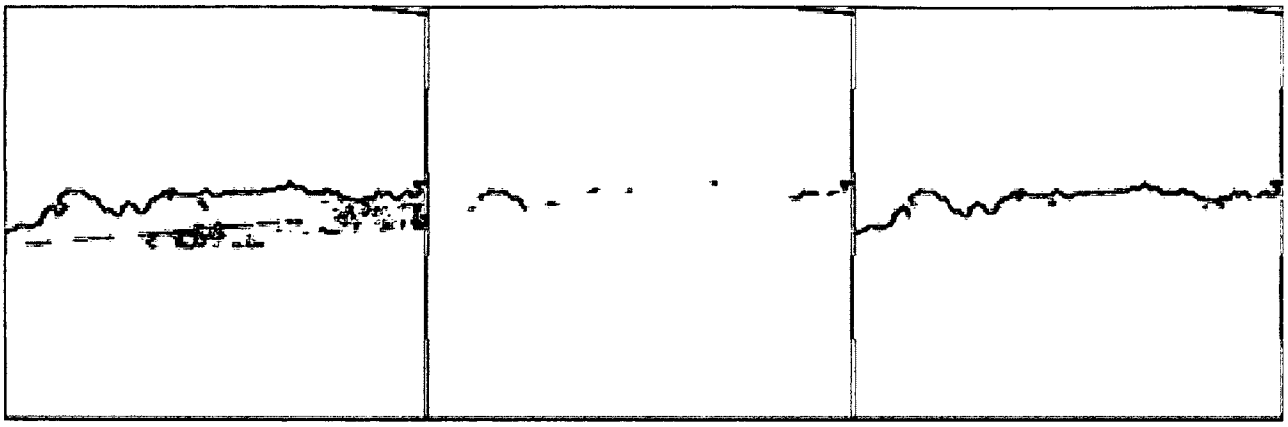

Figure 50 – Example of an original edge map (left, gradient threshold 32), masked with the sky-mask in Figure 49 (center) and masked using a combination of sky and horizon masks (right). Images are inverted and highlighted in red for clarity.

In light of the obstacle avoidance requirements the obstacle detection simply aggregates pixels lying on and in close proximity on the same disparity plane into 2D bounding rectangles. The information passed to the obstacle avoidance algorithm are the 3D locations of the corners of each detected bounding rectangle. These locations are determined by first finding the image co-ordinates of the bounding rectangle corners and then, with the disparity, using the re-projection matrix Q in Equation 41 to determine their 3D locations. For our tests we make an additional assumption to the consistency check in order to remove noise, which is that single disparity pixels with no similar neighbors are excluded from consideration for re-projection. Shown in Figure 51 is the result on the disparity map found in the previous section in sequence #4 with the horizon and sky mask combination applied to remove detail below the horizon. To give a perspective of scale, the extremes (after simple aggregation of any adjacent connected component rectangles) of the rectangular region encompassing obstacle O_1 has corner points (starting top-left and going clockwise) of (204, 16), (254, 16), (254, 177) and (204, 177). These correspond to physical coordinates of (-7.34, -12.96, 45.43), (-4.34, -12.96, 45.43), (-4.34, -3.33, 45.43) and (-7.34, -3.33, 45.43) using a disparity of 11 for the aggregated pixels. This gives the obstacle a width of ap-

proximately 3 meters and a height of 9.63 meters (shortened from the previous estimate due to the horizon).

Figure 51 – Disparity map with similar rectangles grouped by connected components outlined in red (9x9 SAD window, disparity range 1-20, edge gradient threshold of 32)

4.6 Runtime Performance Considerations

All 3 block matching measures were implemented to run on an x86 system in C++ using the Visual Studio environment. In addition each block matching measure was implemented to run on a CUDA GPU. The performance of the non-GPU block matching algorithms is currently poor (although it is un-optimized), with the total runtime of computing both the left and right disparity maps over 16 disparities using 3x3, 5x5 and 7x7 SAD windows being 2, 5.2 and 9 seconds respectively. On the other hand using the custom (un-optimized) GPU version of the SAD block matching algorithm, performance improves considerably to 5.9, 6.34 and 6.68 ms respectively for the respective matching window sizes.

The CUDA GPU based SAD block matching measure currently does not take into account knowledge of edge features. Thus the resulting left and right disparity maps are post processed to

remove disparity information at pixels where there are no edge features (in the process performing the consistency check). The implementation was done this way due to the potential sparseness of edge features in arbitrary scenes, where sparse memory reads and writes could potentially be problematic in terms of performance. The consistency check performed takes 30 ms to complete for the denser ground video sequences and is currently not something which is expected to be easily ported to a GPU since lookups between pixels are not necessarily uniform. Thus in terms of performance it is currently the bottle-neck in the processing pipeline.

4.7 Conclusions

In this chapter an edge feature based approach was proposed for performing obstacle detection, incorporating horizon and sky masks generated by a CNN to remove unwanted detail below the horizon. The resulting feature based image underwent stereo matching, tested with 3 different window correlation techniques, and was found to compute reasonable estimates of the true depth and sizing of potential obstacles which allowed for a 3D model of detected obstacles to be constructed which can be used for collision avoidance.

5 Conclusions

5.1 Summary of Results

The approach to sky and horizon mask generation using a CNN proved to be accurate with the proposed post-processing, namely combining the output from sky and horizon mask networks, generally improving results. The networks were seen to be learning color based filters for discerning the sky and ground regions which was confirmed by testing trained network performance to upside down images. Performance was tested across 2 real-world datasets, achieving training performance similar to that reported in the literature for sky-mask generation using an SVM.

Manual experiments using an SVM confirmed that the performance of trained CNN networks was within 1% of an RBF kernel based SVM. The benefits of using a CNN over an SVM for this task appear to be robustness in the presence of image noise, where trained CNNs were applied to a small, noise and sampling corrupted, dataset achieving testing performance similar to training performance. In additions a significantly smaller amount of memory is required for representing a CNN; something which is handy on resource constrained platforms that may be processing other CNNs which focus on additional tasks such as obstacle detection. It was found that dataset expansion using shifts and rotations was not helpful in increasing network performance, however creating shallow 4 layer networks was seen to provide a balance of good performance and reduced computational requirements.

In order to perform obstacle detection, captured stereo images were rectified and masked to extract salient edge features in the scene. The resulting edge maps were masked using the output from a CNN in order to preserve only edge features at the horizon and above. A SAD window based block matching approach was investigated for performing correspondence of the masked feature images. This approach showed the ability to reasonably estimate the depth and dimensions of potential obstacles in a number of video sequences, allowing a 3D re-projection of pixels aggregated by depth to be made. The final output from the processing pipeline is a set of bounding rectangles located on disparity planes in the scene, and can be used by an obstacle avoidance algorithm to perform aircraft guidance.

5.2 Future Research

In this section possible directions which should merit investigation as a natural continuation of this thesis are presented.

5.2.1 Horizon Detection

To increase the accuracy and possibly relax the heuristic post-processing of the sky-mask, a small committee of CNNs could potentially be trained using different color models for the input images. This is based on the idea that several weak learners should aggregate to produce a stronger learner. In fact a mixture with a competing model such as an SVM could also be a possibility. The feasibility of using a committee of CNNs depends on an evaluation of CNN performance on slower platforms (x86) or possibly an optimization of the existing GPU forward propagation code. This is because the horizon detection stage should really be a starting point in the situations where it's needed and should have minimal impact on any processing pipeline.

Finally in the case where only a horizon line needs to be extracted, one could use the dominant activation region produced by the horizon-mask CNN to infer the best horizon line to draw in the image. This represents a significant reduction in the search space for regions to place possible dividing lines, which in general is anywhere in the input image or anywhere a generic edge detector produces output.

5.2.2 Obstacle Detection

In Section 4 an edge based stereo correspondence algorithm was presented. This algorithm was crafted to exploit the unique appearance of the sky region above the horizon (as compared to the region below) which allows potential obstacles to stand out by large contrast variation. The main idea to exploit this principle was to use edge detection, which essentially became the core obstacle detector. Although shown to work well in principle, the general nature of the edge detectors used raises some question over their sole use for the obstacle detection task, since cloud cover as well as lighting changes and camera bias variations can potentially interfere to produce spurious results. Although edge detection with an adaptive threshold could be used we consider the case

where a CNN could potentially be used as a replacement feature detector. Indeed the general nature of generic edge detectors was the reason why CNNs were employed for the horizon detection task in the first place, sacrificing generality for the ability to learn specialized spatial filters suited for the task.

The success of CNNs for the broad case of detecting 2 obstacles (sky and ground) seems to imply that CNNs could also be suited for general obstacle detection above the horizon. The principle would be the same, requiring the network to produce activations at only the points which pertain to a potential obstacle; the CNN would effectively be acting as an obstacle silhouette detector. A significant body of literature has explored CNNs as general object recognition machines [68][95][96]. These results however typically focus on the labeling of objects into classes where a fully connected layer serves to identify the object in the input. In our case however, the interest is particularly in silhouette detection. The recognition stage could come later in such a system and possibly serve a simple decision system which could take certain action based on actually classifying the detected silhouette; if we've detected an electrical tower there should be wires around and banking left or right might not make sense unless we gain altitude.

Although constructing datasets for training such networks using real-world images seems like a daunting task, it is made easier by the fact that there are numerous datasets already available including CIFAR [97] and NORB [98]; in particular the images with an "airplane" label in each dataset. The problem is that these images don't specify the silhouette for objects. To this end a data generation process could be applied using existing flight videos. If we generate a 3D model of an airplane object, rotate it to some attitude and texture map the obstacle so as to appear real we could then artificially project it into the scene. Having knowledge of the aircraft motion through the MicroPilot datalog would give us information on the ego-motion of the aircraft on

which the original video was taken, allowing us to simulate an approaching obstacle growing in size or even having a more complicated trajectory. Finally, since the object would be artificially generated the ground truth silhouette labeling would be known, allowing a large training dataset to be created.

5.2.3 Stereoscopic Vision

Although capturing clean stereo data is difficult on its own, it is even more difficult to construct ground truth stereo datasets in an outdoor environment. To overcome this problem the distance between the camera and object references frames must be determined. A potential way of obtaining the ground truth distance is to log the GPS coordinates of both reference frames during a test aircraft's approaches towards an object; this is assuming the aircraft is flying at the same altitude, directly towards the obstacle. The MicroPilot autopilot on the Telemaster aircraft has the ability to log GPS position and timestamps while in the case of the test obstacle, a miniature GPS logger such as the TrackStick could be used [99].

In the future, the stereoscopic processing pipeline will need to be expanded to process both the sky and ground regions. This will not mean that horizon detection won't be required; on the contrary, the search process can be tuned to take advantage of the properties of both regions. For the ground region sparse feature extraction, for example using SIFT, SURF or FAST, could be employed to ensure as small a computational burden as possible. It may also be necessary to drop the idea of pre-calibrated stereo all-together, relying instead on computing the fundamental matrix from tracked feature points. This is apparent not just in terms of potential robustness but also since there is additional ongoing research on the GeoSurv project dealing with monocular obstacle detection above and below the horizon as well as feature tracking in order to perform terrain

mapping. Re-using this information for calibration purposes should therefore be a natural future integration step.

In order to improve the robustness of the stereo correspondence stage a more complex sparse correspondence algorithm could be used; one which may also be robust to below the horizon stereo. In particular, belief propagation has been shown to work well for sparse correspondence [73]. Although this approach is typically computationally complex, recent improvements to the core algorithm have shown significant speedup, some of which is particularly tailored to wide baseline stereo vision [74].

To summarize, the contributions of this thesis are:

- The construction of a ground truth, flight captured, image dataset for evaluating the performance of potential horizon detection algorithms.
- Proposing a new approach to performing sky & ground and horizon boundary detection at a high rate of accuracy. The approach involves using convolutional neural networks (the first application of these networks to this task) to automatically learn spatial feature extractors for finding the sky and ground.
- Proposing an above the horizon obstacle detection pipeline based on edge detection and stereoscopic vision which allows detected objects to be localized and mapped in 3D.

References

[1] J. A. B. M. L. Barnard. "The Use of Unmanned Aircraft In Oil, Gas And Mineral E+P Activities." In *Society of Exploration Geophysicists*, page 1132. Las Vegas, Nevada (2008).

[2] M. De Biasio, T. Arnold, R. Leitner, and R. Meestert. "UAV based multi-spectral imaging system for environmental monitoring." Proceedings 2011 20th IMEKO TC2 Symposium on Photonics in Measurement (20th ISPM 2011), pages 69-73. 2011.

[3] D. Casbeer, R. Beard, R. Mehra, and T. McLain. "Forest fire monitoring with multiple small UAVs." Proceedings of the 2005, American Control Conference, pp. 3530-3535, 2005.

[4] R. Girard, A. Howell, and J. Hedrick. "Border patrol and surveillance missions using multiple unmanned air vehicles." 2004 43rd IEEE Conference on Decision and Control (CDC), pp. 620-625, Vol.1, 2004.

[5] Z. Sarris, "Survey of UAV applications in civil markets." In The 9 th IEEE Mediterranean Conference on Control and Automation (MED'01). 2001.

[6] A. Holtzmann, "GeoSurv II Requirement Refinements", Revision 8, Unmanned Aerial Systems Project Group, Carleton University. Available: Carleton UAS Project FTP Server. November 11, 2011.

[7] T. James, "GeoSurv II System Requirements Document". Revision E. Unmanned Aerial Systems Project Group, Carleton University. Available: Carleton UAS Project FTP Server. March 2008.

[8] R. Caron, "Aeromagnetic Surveying Using a Simulated Unmanned Aircraft System", M.A.Sc. Thesis, Carleton University, Available: Carleton UAS Project FTP Server. 2011.

[9] R. Moore, S. Thurrowgood, D. Bland, D. Soccol, and M. Srinivasan. "A stereo vision system for UAV guidance." In *Intelligent Robots and Systems, IROS 2009. IEEE/RSJ International Conference on*, pp. 3386-3391.

[10] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy. "Stereo vision and laser odometry for autonomous helicopters in GPS-denied indoor environments." SPIE, 2009.

[11] J. Byrne, M. Cosgrove, and R. Mehra. "Stereo based obstacle detection for an unmanned air vehicle." In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 2830-2835. 2006.

[12] S. Hrabar, "3D path planning and stereo-based obstacle avoidance for rotorcraft UAVs." In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 807-814. 2008.

[13] B. Call, R. Beard, C. Taylor, and B. Barber. "Obstacle avoidance for unmanned air vehicles using image feature tracking." In *AIAA Guidance, Navigation, and Control Conference*, pp. 3406-3414. 2006.

[14] A. Ortiz, N. Neogi. "Color optic flow: a computer vision approach for object detection on UAVs." In *25th Digital Avionics Systems Conference, 2006 IEEE/AIAA*, pp. 1-12. 2006.

[15] D. Lee, P. Merrell, Z. Wei, and B. Nelson. "Two-frame structure from motion using optical flow probability distributions for unmanned air vehicle obstacle avoidance." *Machine Vision and Applications* 21, no. 3 (2010): 229-240.

[16] C. Yuan, F. Recktenwald, and H. Mallot. "Visual steering of uav in unknown environments." In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 3906-3911. 2009.

[17] D. Lee, R. Beard, P. Merrell, and P. Zhan. "See and avoidance behaviors for autonomous navigation." In *Proceedings of SPIE*, vol. 5609, pp. 23-34. 2004.

[18] S. Hrabar, G. Sukhatme, P. Corke, K. Usher, and J. Roberts. "Combined optic-flow and stereo-based navigation of urban canyons for a UAV." In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 3309-3316. 2005.

[19] S. Griffiths, J. Saunders, A. Curtis, B. Barber, T. McLain, and R. Beard. "Obstacle and terrain avoidance for miniature aerial vehicles." *Advances in Unmanned Aerial Vehicles* (2007): 213-244.

[20] J. Saunders, B. Call, A. Curtis, R. Beard, and T. McLain. "Static and dynamic obstacle avoidance in miniature air vehicles." In *Proc. Infotech@ Aerospace Conf.* 2005.

[21] Geyer, Mark S., and Eric N. Johnson. "3D Obstacle Avoidance in Adversarial Environments for Unmanned Aerial Vehicles." (2006).

[22] S. Hrabar. "An evaluation of stereo and laser-based range sensing for rotorcraft unmanned aerial vehicle obstacle avoidance." *Journal of Field Robotics* (2012).

[23] S. Scherer, S. Singh, L. Chamberlain, and S. Saripalli. "Flying fast and low among obstacles." In *Robotics and Automation, 2007 IEEE International Conference on*, pp. 2023-2029. 2007.

[24] A. Viquerat, L. Blackhall, A. Reid, S. Sukkarieh, and G. Brooker. "Reactive collision avoidance for unmanned aerial vehicles using doppler radar." In *Field and Service Robotics*, pp. 245-254. Springer Berlin/Heidelberg, 2008.

[25] Y. Shen, D. Krusienski, J. Li, Z. Rahman, A Hierarchical Horizon Detection Algorithm, In *IEEE Geoscience and Remote Sensing Letters*, Vol. 10, No.1, pp.111-114, Jan. 2013.

[26] S. Thurrowgood, D. Soccol, R. Moore, D. Bland, and M. V. Srinivasan. "A vision based system for attitude estimation of UAVs." In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 5725-5730. 2009.

[27] D. Dusha, W. Boles, and R. Walker. "Attitude estimation for a fixed-wing aircraft using horizon detection and optical flow." In *Digital Image Computing Techniques and Applications, 9th Biennial Conference of the Australian Pattern Recognition Society on*, pp. 485-492. 2007.

[28] P. Shinzato, V. Grassi, F. Osorio, and D. Wolf. "Fast visual road recognition and horizon detection using multiple artificial neural networks." In *Intelligent Vehicles Symposium (IV)*, pp. 1090-1095. 2012.

[29] B. Zafarifar, and H. Weda. "Horizon detection based on sky-color and edge features." In *Visual Communications and Image Processing 2008*, Vol. 6882, pp. 20-28. 2008.

[30] S. Fefilatyev, V. Smarodzinava, L. Hall, and D. Goldgof. "Horizon detection using machine learning techniques." In *Machine Learning and Applications, 2006. ICMLA'06. 5th International Conference on*, pp. 17-21. 2006.

[31] C. Minwalla, K. Watters, P. Thomas, R. Hornsey, K. Ellis, and S. Jennings. "Horizon extraction in an optical collision avoidance sensor." In *Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference on Electric and Computer Engineering*, pp. 210-214.

[32] T. McGee, R. Sengupta, and K. Hedrick. "Obstacle detection for small autonomous aircraft using sky segmentation." In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 4679-4684. 2005.

[33] N. Boroujeni, A. Etemad, and A. Whitehead. "Robust Horizon Detection using Segmentation for UAV Applications." In *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, pp. 346-352. 2012.

[34] G. de Croon, De Wagter, B. Remes, and R. Ruijsink. "Sky Segmentation Approach to obstacle avoidance." In *Aerospace Conference, 2011 IEEE*, pp. 1-16. 2011.

[35] S. Todorovic, M. Nechyba, and P. Ifju. "Sky/ground modeling for autonomous MAV flight." In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 1, pp. 1422-1427. 2003.

[36] G. Bao, S. Xiong, and Z. Zhou. "Vision-based horizon extraction for micro air vehicle flight control." *Instrumentation and Measurement, IEEE Transactions on* 54, no. 3 (2005): 1067-1072.

[37] S. Ettinger, M. Nechyba, P. Ifju, and M. Waszak. "Vision-guided flight stability and control for micro air vehicles." In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3, pp. 2134-2140. 2002.

[38] R. Duda and P. Hart. "Use of the Hough transformation to detect lines and curves in pictures." *Communications of the ACM* 15, no. 1 (1972): 11-15.

[39] D. Gabor, "Theory of communication. Part 1: The analysis of information." *Electrical Engineers-Part III: Radio and Communication Engineering, Journal of the Institution of* 93, no. 26 (1946): 429-441.

[40] C. Cortes and V. Vapnik. "Support-vector networks." *Machine learning* 20, no. 3 (1995): 273-297.

[41] B. Boser, I. Guyon and V. Vapnik, "*A training algorithm for optimal margin classifiers*". In Haussler, David (editor); 5th Annual ACM Workshop on COLT, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

[42] B. Scholkopf, C. Burges, and V. Vapnik, "*Incorporating invariances in support vector learning machines*". In *Artificial Neural Networks —ICANN'96*, pages 47 – 52, Berlin, 1996.

[43] V. Blanz, V. Scholkopf, B. Bulthoff, C. Burges, V. Vapnik, and T. Vetter, "*Comparison of view–based object recognition algorithms using realistic 3d models*". In *Artificial Neural Networks—ICANN'96*, pages 251 – 256, Berlin, 1996.

[44] E. Osuna, R. Freund, and F. Girosi, "*Training support vector machines: an application to face detection*". In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 130 – 136, 1997.

[45] T. Joachims, "Svmlight: Support vector machine." *SVM-Light Support Vector Machine* http://svmlight.joachims.org/, *University of Dortmund* 19, 1999.

[46] C. Chang, and C. Lin, "LIBSVM: a library for support vector machines." *ACM Transactions on Intelligent Systems and Technology (TIST)*. 2011.

[47] T. Joachims, "Making large scale SVM learning practical." (1999).

[48] J. Platt. "Sequential minimal optimization: A fast algorithm for training support vector machines." (1998).

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient based learning applied to document recognition". *Proceedings of the IEEE*, 86(11):2278–2324, November 1998

[50] K. Fukushima, and S. Miyake. "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position." *Pattern recognition* 15, no. 6 (1982): 455-469.

[51] J. Mutch, and D. Lowe. "Multiclass object recognition with sparse, localized features." In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1, pp. 11-18. 2006.

[52] T. Serre, L. Wolf, and T. Poggio. "Object recognition with features inspired by visual cortex." In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 994-1000. 2005.

[53] D. Hubel, and T. Wiesel. "Receptive Fields, Binocular Interaction And Functional Architecture In The Cat's Visual Cortex", Journal of Physiology, 160, pp. 106–154. 1962.

[54] P. Simard, D. Steinkraus, and J. Platt. "Best Practice for Convolutional Neural Networks Applied to Visual Document Analysis". *In International Conference on Document Analysis and Recogntion (ICDAR)*, IEEE Computer Society, Los Alamitos, pages 958{962, 2003.

[55] C. Bishop, "Pattern Recognition and Machine Learning", Springer, 2006.

[56] M. Hagan, and M. Menhaj. "Training feedforward networks with the Marquardt algorithm." *Neural Networks, IEEE Transactions on* 5, no. 6 pp. 989-993. 1994.

[57] J. More, "The Levenberg-Marquardt algorithm: implementation and theory."*Numerical analysis,* pp. 105-116. 1978.

[58] C. Cortes, and M. Mohri. "AUC optimization vs. error rate minimization." *Advances in neural information processing systems* 16, no. 16, pp. 313-320. 2004.

[59] M. Osadchy, Y. LeCun, and M. Miller. "Synergistic Face Detection and Pose Estimation with Energy-Based Models". *Journal of Machine Learning Research*, 8:1197-1215, May 2007.

[60] D. Rumelhart, G. Hinton, and R. Williams. "Learning representations by back-propagating errors." *Nature* 323, no. 6088 (1986): 533-536.

[61] D. Scherer, A. Müller, and S. Behnke. "Evaluation of pooling operations in convolutional architectures for object recognition." *Artificial Neural Networks–ICANN 2010* (2010): 92-101.

[62] Boureau, Y-Lan, Jean Ponce, and Yann LeCun. "A theoretical analysis of feature pooling in visual recognition." In *International Conference on Machine Learning*, pp. 111-118. 2010.

[63] Zhang, Z., Shan, Y., 2000. A progressive scheme for stereo matching. Proc. 2nd Workshop Structure from Multiple Image of Large Scale Environments, ECCV Workshop, July 2000, pp. 26–31.

[64] D. Scharstein and R. Szeliski. "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms." *International journal of computer vision* 47, no. 1 (2002): 7-42.

[65] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun. "Cnp: An fpga-based processor for convolutional networks." In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pp. 32-37. 2009.

[66] J. Cloutier, E. Cosatto, S. Pigeon, F. Boyer and P.Y. Simar, "Vip: An fpga-based processor for image processing and neural networks," in *Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems (MicroNeuro '96)*, Lausanne, Switzerland, 1996, pp. 330-336.

[67] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi. "A dynamically configurable coprocessor for convolutional neural networks." *ACM SIGARCH Computer Architecture News* 38, no. 3 (2010): 247-257.

[68] D. Cireşan, U. Meier, J. Masci, L. Gambardella, and J. Schmidhuber. "Flexible, high performance convolutional neural networks for image classification." In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, pp. 1237-1242. 2011.

[69] D. Scherer, H. Schulz, and S. Behnke. "Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors." *Artificial Neural Networks–ICANN 2010* (2010): 82-91.

[70] F. Nasse, C. Thurau, and G. Fink. "Face detection using gpu-based convolutional neural networks." In *Computer Analysis of Images and Patterns*, pp. 83-90. Springer Berlin/Heidelberg, 2009.

[71] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello. "Hardware accelerated convolutional neural networks for synthetic vision systems." In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pp. 257-260. 2010.

[72] D. Strigl, K. Kofler, and S. Podlipnig. "Performance and scalability of GPU-based convolutional neural networks." In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pp. 317-324. 2010.

[73] M. Sarkis, and K. Diepold. "Sparse stereo matching using belief propagation." *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on.* 2008.

[74] J. Coughlan, H. Shen. "An Embarrassingly Simple Speed-Up of Belief Propagation with Robust Potentials." *arXiv preprint arXiv:1010.0012*(2010).

[75] B. Catanzaro, N. Sundaram, and K. Keutzer. "Fast support vector machine training and classification on graphics processors." In*Proceedings of the 25th international conference on Machine learning*, pp. 104-111. ACM, 2008.

[76] J. Banks, M. Bennamoun, and P. Corke. "Non-parametric techniques for fast and ro bust stereo matching." In *TENCON'97. IEEE Region 10 Annual Conference. Speech and Image Techno logies for Computing and Telecommunications., Proceedings of IEEE*, vol. 1, pp. 365-368. 1997.

[77] H. Hirschmuller, and D. Scharstein. "Evaluation of stereo matching costs on images with radiometric differences." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 31, no. 9 (2009): 1582-1599.

[78] S. Birchfield, and C. Tomasi. "A pixel dissimilarity measure that is insensitive to image sampling." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 20, no. 4 (1998): 401-406.

[79] T. Dang, C. Hoffmann, and C. Stiller. "Continuous stereo self-calibration by camera parameter tracking." *Image Processing, IEEE Transactions on* 18, no. 7, pp. 1536-1550. 2008.

[80] S. Owlia, "Developing a Method for Real-Time Reactive Obstacle Avoidance for a Low-Altitude Fixed-Wing Aircraft", M.A.Sc. Thesis, 2013.

[81] O. Faugeras, *Three-dimensional computer vision*. MIT press, 1993.

[82] G. Bradski, and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Incorporated, 2008.

[83] D. Lowe, "Object recognition from local scale-invariant features."*Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. Vol. 2. Ieee, 1999.

[84] H. Bay, T. Tuytelaars, and L. Gool. "Surf: Speeded up robust features." *Computer Vision–ECCV 2006* (2006): 404-417.

[85] E. Rosten, and T. Drummond. "Machine learning for high-speed corner detection." *Computer Vision–ECCV 2006* (2006): 430-443.

[86] Szeliski, Richard. *Computer vision: Algorithms and applications*. Springer, 2010.

[87] Y. Boykov, O. Veksler, and R. Zabih. "Fast approximate energy minimization via graph cuts." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 23.11 (2001): 1222-1239.

[88] J. Sun, N. Zheng, and H. Shum. "Stereo matching using belief propagation." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 25.7 (2003): 787-800.

[89] Y. Ohta, and T. Kanade. "Stereo by intra-and inter-scanline search using dynamic programming." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 2 (1985): 139-154.

[90] O. Veksler, "Stereo correspondence by dynamic programming on a tree."*Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 2. IEEE, 2005.

[91] R. Szeliski, and D. Scharstein, "Sampling the disparity space image". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3):419–425. 2004.

[92] D. Ziou and S. Tabbone, "Edge detection techniques: An overview", *International Journal of Pattern Recognition and Image Analysis*, 8(4):537–559, 1998.

[93] Nvidia, C. U. D. A. "Compute unified device architecture programming guide." (2007).

[94] N. Kirsch, "NVIDIA ION – Intel ATOM Done Right?", http://www.legitreviews.com/article/934/12/, Mar. 20, 2009.

[95] C. Farabet, C. Couprie, L. Najman, Y. LeCun, "Learning Hierarchical Features for Scene Labeling", To Appear in *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2013.

[96] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, Y. LeCun, "Learning long-range vision for autonomous off-road driving."*Journal of Field Robotics* 26.2 (2009): 120-144.

[97] A. Krizhevsky, and G. Hinton. "Learning multiple layers of features from tiny images." *Master's thesis, Department of Computer Science, University of Toronto* (2009).

[98] Y. LeCun, F. Huang, and L. Bottou. "Learning methods for generic object recognition with invariance to pose and lighting." *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on.* Vol. 2. IEEE, 2004.

[99] TrackStick GPS Receiver Modules, http://www.trackstick.com/, 2012.

[100] S. Suzuki, "Topological structural analysis of digitized binary images by border following." *Computer Vision, Graphics, and Image Processing* 30.1 (1985): 32-46.

[101] OpenCV Documentation on Structural Analysis and Shape Descriptors, http://opencv.willowgarage.com/documentation/structural_analysis_and_shape_descriptors.html, 2012.

[102] R. Hartley, "In defense of the eight-point algorithm." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 19.6 (1997): 580-593.

[103] T. Clarke, and J. Fryer. "The development of camera calibration methods and models." *The Photogrammetric Record* 16.91 (2003): 51-66.