Virginia Commonwealth University
VCU Scholars Compass

Theses and Dissertations

Graduate School

2019

The Application of Index Based, Region Segmentation, and Deep Learning Approaches to Sensor Fusion for Vegetation Detection

David L. Stone
Virginia Commonwealth University

Follow this and additional works at: https://scholarscompass.vcu.edu/etd

Part of the Electrical and Computer Engineering Commons

© The Author

Downloaded from
https://scholarscompass.vcu.edu/etd/5708

This Dissertation is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.

©David Stone 2019

All Rights Reserved

The Application of Index Based, Region Segmentation, and Deep Learning Approaches to Sensor Fusion for Vegetation Detection

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

By

David L. Stone

O.E, Massachusetts Institute of Technology 1982

M.S.M.E, Massachusetts Institute of Technology 1982

B.S.E.E, Purdue University 1973

Director: Yuichi Motai, Ph.D.,

Associate Professor, Electrical and Computer Engineering

Virginia Commonwealth University

Richmond, Virginia

January, 2019

Acknowledgment

This study was supported in part by the Office of Naval Research, in part by the Naval Surface Warfare Center Dahlgren Division (NSWCDD), in part by the Marine Corps Warfighting Lab, Quantico, VA, and in part by Air Force Research Laboratory, Rome NY. I would also like to acknowledge Adrian Rosebrock and his PyImageSearch Deep Learning book and course for his invaluable guidance in learning deep learning with python.

I would like to express my deepest appreciation to my Ph.D. advisor Dr. Yuichi Motai for the continuous support for my studies and related research, for patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this dissertation. I would like to thank my dissertation committee members, Dr. Ashok Iyer, Dr. Alen Docef, Dr. Ruixin Niu, Dr. Milos Manick, and Dr. Kayvan Najarian for their help and for their direction with this dissertation.

I would like to thank my fellow graduate students Emrah Benli, Seonyeong Park, Nahian Alam Siddique, Ammar Osama for their support during this great period. I would also like to express my appreciation to Gurav Shaw for his help throughout my time at VCU.

Finally, I would like to thank my wife Nancy Stone, my children Michelle, Daniel, John, Eddie, and Eric for their unending love and patience during the past eight years it has taken me to graduate. I would also like to thank my peers at work for their unending support and patience during this period.

Table of Contents

Acknowledgment iii

List of Tables vii

Table of Figures viii

List of Abbreviations x

Abstract xi

1. Chapter - Introduction 1

1.1. Problem Statement 1

1.2. Introduction of Omnidirectional far-infrared Camera System and Calibration 3

1.3. Introduction to Index Based and Region Segmentation Based Vegetation Detection 5

1.4. Introduction to Deep Learning Sensor Fusion 8

1.5. Impact 10

1.6. Innovations 13

1.7. Summary 16

2. Chapter – Prior Related Works 18

2.1. Prior Related Works for Omni-direction Camera Calibration 18

2.2. Prior Related Works for Index and Vegetation Region Segmentation Methods 23

2.3. Prior Related Works for Deep Learning Sensor Fusion 29

2.3.1. Visual and Infrared Index, Histogram, and Region Segmentation Based Vegetation Detection 29

2.3.2. Deep Learning Convolution Neural Network Fusion of O-D IR and O-D Visual Stream 31

3. Chapter – Technical Approach for the Three Methods 35

3.1. Technical Approach for Direct Spherical Calibration of Omnidirectional Far Infrared Camera System35

3.1.1. Omnidirectional IR camera geometry 35

3.1.2. DSC omnidirectional IR camera calibration methodology 37

3.1.3. Optimal number selection of image capture of calibration boards 46

3.1.4. Comparison to other methods - Baselines 1,2,3 49

3.2. Technical Approach for Index and Region Segmentation for Sensor Fusion of Omnidirectional Far-infrared Camera and Visual Camera for Vegetation Detection 51

3.2.1. Omni-direction Camera Setting 52

3.2.2. Visual and IR Index Based Vegetation Detection 54

3.2.3. IR Stream Segmentation using Region based Thermal Analysis 59

3.2.4. Genetic Algorithm 63
3.2.5. Thermal Region Fusion 63
3.3. Technical Approach for Vegetation Detection through Deep Learning Sensor Fusion of Omnidirectional (O-D) Far-infrared and O-D Visual Stream 68
3.3.1. Overview of Technical approach for Deep Sensor Fusion 68
3.3.2. Omni-direction Camera Setting 71
3.3.3. Visual and Infrared Index, Histogram, and Segmentation Based Vegetation Detection 71
3.3.4. Autoencoder - Convolution Neural Network 73
3.3.4.1. Auto-encoder Neural Network 74
3.3.4.2. Sparse Auto-encoder 76
3.3.5. Convolution Neural Network (CNN) 78
3.3.6. Deep Learning Fusion Network (DeepFuseNet) 81
3.3.7. Validation Methodologies 84
3.3.8. Visulization of Network Filters 85
4. Chapter – Experimental Results for the Three Methods 88
4.1. Direct Spherical Calibration Experimental Results 88
4.1.1. Calibration data setting for evaluation 88
4.1.2. DSC calibration 91
4.1.3. Comparison to other Baseline calibration methods 93
4.1.4. Error analysis of four methods 96
4.1.5. Optimal number selection of image capture of calibration boards 99
4.2. Index Based and Thermal Region Fusion Experimental Results 105
4.2.1. Omni-direction Camera and Data Setting 105
4.2.2. Visual and IR Index Based Vegetation Detection 107
4.2.3. IR Stream Segmentation using Region based Thermal Analysis 109
4.3. DeepFuseNet Experimental Results 114
4.3.1. Omni-direction Camera Setting 114
4.3.2. Visual and Infrared MNDVI and TRF Vegetation Detection 115
4.3.3. Autoencoder - Convolution Neural Network Fusion of O-D IR and Visual Stream 116
4.3.4. Convolution Neural Net (CNN) 118
4.3.5. DeepFuseNet 123
5. Chapter – Conclusion 126

5.1. Direct Spherical Calibration (DSC) Conclusion 126
5.2. Index Based and Thermal Region Fusion Methods Conclusion 127
5.3. DeepFuseNet Conclusion 128
5.4. Summary Conclusion 128
APPENDIX A 132
REPRESENTATIVE MATLAB AND PYTHON CODES 132
A.1 Calibration GUI (omni_calib_gui_normal_s.m) 133
A.2 Go_omni_calib (go_omni_calib_itr_s.m) 136
A.3 Spherical Coordinates (Spherical_s.m) 161
A.4 Multi-objective Optimization () 163
A.5 Compute Omnidirectional Error (comp_omni_error_s.m) 163
A.6 Analyse Error (analyse_err_s.m) 165
A.7 Vegetation Detection IR Region (vegdetect_IRR.m) 170
A.8 IR Region (IR_Region.m) 179
A.9 Genetic Algorithm Clustering (GACluster.m) 180
A.10 Clustering Cost Function (ClusteringCost2.m) 183
A11. Bottleneck Feature Extractor (veg_bottleneck.py) 184
A12. Autoencoder Feature Extractor 188
A.13 Vegetation Finetuning (veg_finetune.py) 191
A14. Deep Fusion Network (DeepFuseNet_plants18.py) 193
References 206

List of Tables

Table 1. Camera Settings 8
Table 2. Significance, Innovation and Technical Merits 15
Table 3. Methodology Summary 19
Table 4. Omnidirectional (O-D) Camera Applications Summary 24
Table 5. Vegetation Detection Methodology Summary 25
Table 6. Segmentation and Histogram Method Summary 26
Table 7. Sensor Fusion Method Summary 27
Table 8. Vegetation Detection Methodology Summary 30
Table 9. Histogram and Segmentation Method Summary 31
Table 10. Neural Network and Deep Learning Method Summary 31
Table 11. Relationship Between Eccentricity e and Mirror Parameter ξ 41
Table 12. Five Process Steps for DSC 45
Table 13. Methodology Comparison 49
Table 14. Proposed TRF Algorithm: Sensor Fusion of MNDVI Vegetation Index and Region Based IR 64
Table 15. Deep Fusion Network for o-d ir and visual stream 68
Table 16. Calibration Image Sets 89
Table 17. Methodology Numerical Comparison 99
Table 18. Ratio Calibrated to Actual(Rca) Comparison 101
Table 19. Optimality Comparison 104
Table 20. Camera Data Settings 105
Table 21. Thermal Segmented Region Fusion Results - Comparison of MNDVI to TRF 110
Table 22. Other Method Comparison Results 111
Table 23. Camera Data Settings 114
Table 24. Baseline and DeepFuseNet Results 124
Table 25. Summary of Methodologies 129

Table of Figures

Figure 1. Omnidirectional IR calibration images 5
Figure 2. Robot and Camera Setting 6
Figure 3. Robot with O-D IR, O-D Visual, and Kinnect Cameras 10
Figure 4. Representative depiction of three goals 15
Figure 5. Concept level depiction of Deep Learning Training and DeepFuseNet deployment ... 17
Figure 6. Omnidirectional Camera Geometry 36
Figure 7. Omnidirectional Geometry Relationship 37
Figure 8. Camera Calibration Overview 38
Figure 9. Camera Calibration setup: 39
Figure 10. Spherical Coordinate Diagram: 42
Figure 11. Calibration Board Capture 44
Figure 12. Pareto Optimization Diagram: 47
Figure 13. Ratio Calibrated to Actual: 48
Figure 14. Baseline Fusion Block Diagram 51
Figure 15. Geometry Relationship 54
Figure 16. Spherical-Cartesian Coordinates 56
Figure 17. MNDVI Approach 57
Figure 18. Cumulative Density Function 58
Figure 19. IR Region Segmentation 60
Figure 21. MNDVI and TRF Comparison O-D and Kinect 65
Figure 22. MNDVI and TRF Comparison O-D Cameras 67
Figure 23. Deep Fusion Network Block Diagram 70
Figure 24. MNDVI False Detects 72
Figure 25. Autoencoder Layers 75
Figure 26. Autoencoder Fusion 78
Figure 27. Merged Model Functional Block Diagram 83
Figure 28. Deep Fusion Network (DeepFuseNet) 84
Figure 29. Convolution Layer Feature Map Visualization 86
Figure 30. Activation Filter Map Visualization 87
Figure 31. Robot and Camera Setup: 89
Figure 32. Calibration Re-projection points: 90
Figure 33. Extrinsic Mapping: 91
Figure 34. Sparse Data Set Error 92
Figure 35. Moderate Data Set Error 93
Figure 36. Dense Data Set Error 95
Figure 37. Error Scatter Plot 96
Figure 38. Error vs Radius Plot 97
Figure 39. Comparison Polar Plot 100

Figure 40. Ratio Calibrated to Actual (R_{CA}) 102
Figure 41. Methods - Error Bar Chart 103
Figure 42. Pareto Optimization Plot 104
Figure 43. RGB and IR Histogram for O-D IR and Kinect Camera 106
Figure 44. RGB and IR Histogram for O-D IR and O-D Visual Cameras 107
Figure 45. O-D IR and Kinnect MNDVI and TRF Comparison 108
Figure 46. O-D IR and O-D Visual MNDVI and TRF Comparison 109
Figure 47. Receiver Operator Curve (ROC) O-D IR and Kinnect 111
Figure 48. Receiver Operator Curve for O-D IR and O-D Visual 112
Figure 49. RGB and IR Histogram 115
Figure 50. TRF Fusion Results 116
Figure 51. VGG16 Autoencoder Accuracy and Loss 117
Figure 52. Autoencoder Fusion Accuracy and Loss 118
Figure 53. ResNet Accuracy and Loss Fine-tuning 119
Figure 54. VGG16 Bottleneck Feature Extractor 120
Figure 56. R5 Accuracy Comparison 121
Figure 57. Cross-Entropy Loss Comparison 122
Figure 58. Fine-tuned Merged Model 123
Figure 59. Vegetation Detection Results 124

List of Abbreviations

2D	**2 Dimensional**
3D	**3 Dimensional**
ANN	**Artificial Neural Network**
CNN	**Convolutional Neural Network**
DL	**Deep Learning**
DLSF	**Deep Learning Sensor Fusion (DeepFuseNet)**
DLT	**Direct Linear Transform**
DOD	**Department of Defense**
DSC	**Direct Spherical Calibration**
FOV	**Field of View**
FPR/TPR	**False Positive Rate / True Positive Rate**
FPGA	**Field-Programmable Gate Array**
HRI	**Human Robot Interaction**
IR	**Infrared**
IRP	**Intelligent Robotic Perception**
KF	**Kalman Filter**
EKF	**Extended Kalman Filter**
Lidar	**Light Imaging, Detection, and Ranging**
MAP	**Maximum A Posteriori**
MNDVI	**Modified Normalized Difference Vegetation Index**
NDVI	**Normalized Difference Vegetation Index**
O-D	**Omni-Directional**
ONR	**Office of Naval Research**
SURF	**Speeded Up Robust Features**
SFM	**Structure from Motion**
RGB	**Red Green Blue**
RMSE	**Normalized Root Mean Squared Error**
RCTA	**Robotics Collaborative Technology Alliance**
STD	**Standard Deviation**
SUMET	**Small Unit Mobility Enhancement Technology**
SVD	**Singular Value Decomposition**
TRF	**Thermal Region Fusion**
UGV	**Unmanned Ground Vehicle**
UCR	**Ubiquitous Collaborative Robots**

Abstract

This thesis investigates the application of index based, region segmentation, and deep learning methods to the sensor fusion of omnidirectional (O-D) Infrared (IR) sensors, Kinnect sensors, and O-D vision sensors to increase the level of intelligent perception for unmanned robotic platforms. The goals of this work is first to provide a more robust calibration approach and improve the calibration of low resolution and noisy IR O-D cameras. Then our goal was to explore the best approach to sensor fusion for vegetation detection. We looked at index based, region segmentation, and deep learning methods and compared them with a goal of significant reduction in false positives while maintaining reasonable vegetation detection.

The results are as follows:

- Direct Spherical Calibration of the IR camera provided a more consistent and robust calibration board capture and resulted in the best overall calibration results with sub-pixel accuracy
- The best approach for sensor fusion for vegetation detection was the deep learning approach, the three methods are detailed in the following chapters with the results summarized here.
    - Modified Normalized Difference Vegetation Index approach achieved 86.74% recognition and 32.5% false positive, with peaks to 80%
    - Thermal Region Fusion (TRF) achieved a lower recognition rate at 75.16% but reduced false positives to 11.75% (a 64% reduction)
    - Our Deep Learning Fusion Network (DeepFuseNet) results demonstrated that deep learning approach showed the best results with a significant (92%) reduction

in false positives when compared to our modified normalized difference vegetation index approach. The recognition was 95.6% with 2% false positive.

Current approaches are primarily focused on O-D color vision for localization, mapping, and tracking and do not adequately address the application of these sensors to vegetation detection. We will demonstrate the contradiction between current approaches and our deep sensor fusion (DeepFuseNet) for vegetation detection. The combination of O-D IR and O-D color vision coupled with deep learning for the extraction of vegetation material type, has great potential for robot perception. This thesis will look at two architectures: 1) the application of Autoencoders Feature Extractors feeding a deep Convolution Neural Network (CNN) fusion network (DeepFuseNet), and 2) Bottleneck CNN feature extractors feeding a deep CNN fusion network (DeepFuseNet) for the fusion of O-D IR and O-D visual sensors. We show that the vegetation recognition rate and the number of false detects inherent in the classical indices based spectral decomposition are greatly improved using our DeepFuseNet architecture.

We first investigate the calibration of omnidirectional infrared (IR) camera for intelligent perception applications. The low resolution omnidirectional (O-D) IR image edge boundaries are not as sharp as with color vision cameras, and as a result, the standard calibration methods were harder to use and less accurate with the low definition of the omnidirectional IR camera. In order to more fully address omnidirectional IR camera calibration, we propose a new calibration grid center coordinates control point discovery methodology and a Direct Spherical Calibration (DSC) approach for a more robust and accurate method of calibration. DSC addresses the limitations of the existing methods by using the spherical coordinates of the centroid of the calibration board to directly triangulate the location of the camera center and iteratively solve for the camera parameters. We compare DSC to three Baseline visual calibration methodologies and augment them with additional output of the spherical results for comparison. We also look at the optimum number of calibration boards using an evolutionary algorithm and Pareto optimization to find the best method and combination of accuracy, methodology and number of calibration boards. The benefits of DSC are more efficient calibration board geometry selection, and better accuracy than the three Baseline visual calibration methodologies.

In the context of vegetation detection, the fusion of omnidirectional (O-D) Infrared (IR) and color vision sensors may increase the level of vegetation perception for unmanned robotic platforms. A literature search found no significant research in our area of interest. The fusion of O-D IR and O-D color vision sensors for the extraction of feature material type has not been adequately addressed. We will look at augmenting indices based spectral decomposition with IR region based spectral decomposition to address the number of false detects inherent in indices based spectral decomposition alone. Our work shows that the fusion of the Normalized Difference Vegetation Index (NDVI) from the O-D color camera fused with the IR thresholded signature region associated with the vegetation region, minimizes the number of false detects seen with NDVI alone. The contribution of this work is the demonstration of two new techniques, Thresholded Region Fusion (TRF) technique for the fusion of O-D IR and O-D Color. We also look at the Kinect vision sensor fused with the O-D IR camera. Our experimental validation demonstrates a 64% reduction in false detects in our method compared to classical indices based detection.

We finally compare our DeepFuseNet results with our previous work with Normalized Difference Vegetation index (NDVI) and IR region based spectral fusion. This current work shows that the fusion of the O-D IR and O-D visual streams utilizing our DeepFuseNet deep learning approach out performs the previous NVDI fused with far infrared region segmentation. Our

experimental validation demonstrates an 92% reduction in false detects in our method compared to classical indices based detection. This work contributes a new technique for the fusion of O-D vision and O-D IR sensors using two deep CNN feature extractors feeding into a fully connected CNN Network (DeepFuseNet).

1. Chapter - Introduction

Accurate estimation of the materials in a robot agent's (agent) environment is crucial to Intelligent Robotic Perception (IRP). Without the recognition of the semantic structure of the scene, the agent will not be able to adequately interact with the world or the people in it. Many existing techniques handle object detection by placing walls or non-passable blocks in the world map when the sensors detect something in the environment [1]. Others place color maps representing the height of obstacles and thus identifies the area as non-passable [2]. These approaches have a major drawback as they do not address the density of or pass-ability of the materials in the environment. An example is when the agent is following its human counterpart and the human walks into tall grass. The human recognizes it as grass and that it is passable. The current state of robot perception is that the agent sees it as a wall and goes a different less efficient route. Other methods attempt to address this by several methods of vegetation detection, but these have a lot of false detects and can't detect dry grass [3].

1.1. Problem Statement

In robotic platform development, there is a need for overcoming perception issues and applying robust cognitive behaviors to make the use of robotic platforms more practical. While there has been a great deal of research in the area of intelligent robotic perception, the practical application of Unmanned Ground Vehicles (UGV) to operation with small teams of humans in many civilian, service, and military applications is limited by the current state of intelligent perception and the high cost of sensors. The current state of obstacle detection and avoidance is able to handle well-structured obstacles, but is limited in distinguishing between solid obstacles and low density passable objects. We are leveraging the benefits of potentially low cost Omni-Directional (O-D) infrared (IR) and color vision sensors coupled with intelligent perception algorithms to better address this perceptual framework. O-D far-IR is required to augment the visual scene perception by adding a world view of the thermal characteristics of the materials in the robot's environment. The thermal properties will be fused with the visual color and texture properties to better characterize the scene materials semantic structure.

Thus, our goals in this research are to identify the spectral and thermal signature of materials in the environment, and to build a semantic overlay model that better characterizes the scene. For these, we pursue the following technical merits.

1) By introducing a Direct Spherical Calibration (DSC) method to address calibration of the omnidirectional (O-D) far infrared camera which will be used to provide thermal region data for our algorithms. We make a significant improvement in the reliability of calibration board capture in the low resolution and noisy IR O-D camera. We also significantly reduce the error to sub-pixel accuracy.

2) By introducing the scene semantic feature extraction algorithm, more precise material characteristics such as spectral signature, thermal properties, color and texture can be learned and extracted to better characterize the agent's scene.

3) By reconstructing the semantic scene through deep learning artificial neural network fusion, the proposed novel study can offer valuable scene characteristics for the agent to better evaluate its environment. More accurate decisions will be possible by allowing the agent a better understanding of scene semantics due to an advantage of recognition of O-D infrared and visual cues to aid the recognition of materials.

The results are as follows:

- Direct Spherical Calibration of the IR camera provided a more consistent and robust calibration board capture and resulted in the best overall calibration results with sub-pixel accuracy
- The best approach for sensor fusion for vegetation detection was the deep learning approach, the three methods are detailed in the following chapters with the results summarized here.
    - Modified Normalized Difference Vegetation Index approach achieved 86.74% recognition and 32.5% false positive, with peaks to 80%
    - Thermal Region Fusion (TRF) achieved a lower recognition rate at 75.16% but reduced false positives to 11.75% (a 64% reduction)
    - Our Deep Learning Fusion Network (DeepFuseNet) results demonstrated that deep learning approach showed the best results with a significant (92%) reduction in false positives when compared to our modified normalized difference vegetation index approach. The recognition was 95.6% with 2% false positive.

1.2. Introduction of Omnidirectional far-infrared Camera System and Calibration

The contribution of this work is the development of a new O-D IR camera calibration framework, called Direct Spherical Calibration (DSC). Based on our experimental results, the DSC method fits in IR O-D sensing, because it provides IR O-D camera calibration with:

- A more accurate and robust calibration board geometry selection,
- More direct calibration method, and that it more effectively finds the key projection point for each calibration board with resultant improvement in accuracy (sub pixel), when compared to the existing methods.
- Best fit of ratio of calibrated to actual.

While there has been a great deal of research in the area of camera calibration in general, and omnidirectional (O-D) camera calibration in particular, there has been little published work on the calibration of lower resolution O-D infrared (IR) cameras. We propose a new omnidirectional IR camera calibration framework, called Direct Spherical Calibration (DSC), and we apply a more robust corner selection method to overcome the problems with identifying the control points from the calibration grid in the lower resolution O-D IR camera. The authors are leveraging the benefits of potentially low cost omnidirectional (O-D) infrared (IR) camera and color vision sensors applied to sensing for intelligent robotic perception applications. The motivation for the calibration of the O-D camera is to lay the foundation for our work in vegetation detection, human tracking, and 3D scene reconstruction. The calibration helps us find the center of the camera and more accurately reconstruct the semantic geometry of the scene. Fig. 1 shows an example of the calibration grid captured by the O-D IR camera, and the problem with accurately capturing the grid corner intersections in the low resolution calibration grid can be seen from the lack of clarity in some of the images sampled around the O-D IR camera. All of the baseline methods that we used for comparison used automatic corner selection which was not reliable in the lower resolution IR camera and required the authors to manually correct the baseline method's corner selection results. The poor quality of the grid selection in the lower resolution IR image resulted in higher error and in some cases a failure of the baseline calibration methods to converge. We propose a more effective direct triangulation approach.

The existing omnidirectional camera calibration methods are optimized for visual cameras. The omnidirectional IR image edge boundaries are not as sharp as with color vision cameras, and as a result, the standard calibration methods were harder to use and in some cases failed when applied to the low definition omnidirectional IR camera, and in some cases completely failed.

The traditional approaches use a re-projection method based on unreliable control point resolution in the low resolution IR setting. The motivation for the calibration of the O-D camera is to lay the foundation for our work in vegetation detection, human tracking, and 3D scene reconstruction. The calibration helps us find the center of the camera and more accurately reconstruct the semantic geometry of the scene. Omnidirectional cameras are being applied in computer vision and robotics because of their advantages in wide field of view (FOV). Much work has been done on calibration methods optimized in the visual band but very little in the IR band.

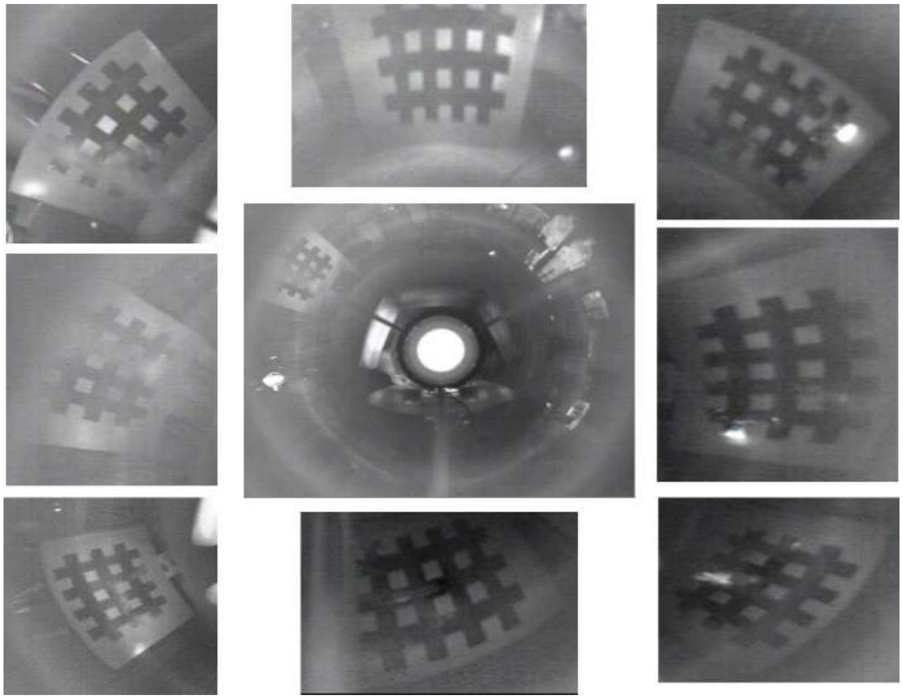


Figure 1. Omnidirectional IR calibration images

O-D camera images with several representative calibration board grids. Due to the low resolution you can see the problem with accurately capturing the calibration control points.

1.3. Introduction to Index Based and Region Segmentation Based Vegetation Detection

While there has been a great deal of research in the area of intelligent perception, the practical application of Omnidirectional (O-D) sensing, both O-D Infrared (IR) and O-D color vision sensors, to vegetation detection has not received adequate attention. This work explores the reduction of false positives in index based vegetation recognition by the fusion of these two sensors. The contribution of this work is the demonstration of a new technique called Thresholded Segmentation Fusion (TRF) to fuse the visual/IR Normalized Difference Vegetation Index (NDVI) based results with the segmented thermal vegetation region from the O-D IR sensor.

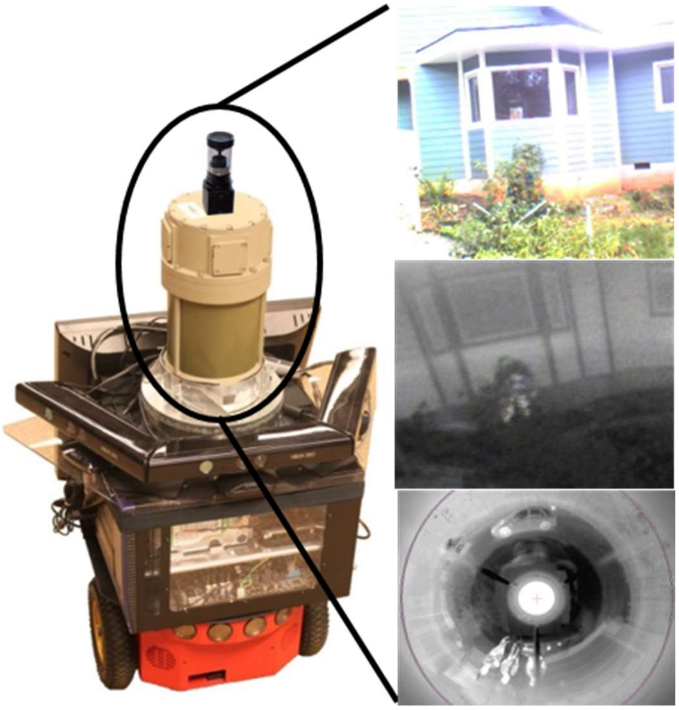


Figure 2. Robot and Camera Setting
O-D IR, O-D Color, and Kinect cameras, and onboard computer. Showing a sample O-D IR image and two extracted IR and Color images of front of the house and a man crouched behind the bush.

The authors are leveraging the benefits of potentially low cost omnidirectional (O-D) infrared (IR) and O-D Color Cameras Fig 2. While we were waiting for our O-D Color camera, we investigated the fusion of the O-D IR camera with the Kinect color vision sensor. These sensors coupled with intelligent perception algorithms may provide the solution of this problem. With improved omnidirectional systems, the corrections required are reduced, compared to multiple camera stitching, to create 360-degree fusion of our Omnidirectional (O-D) IR and O-D Color cameras, the computation requirements are reduced and sources of error are improved for electro-optical cameras. Additionally, multi-spectral sources can be fused with reduced error and the calculation errors for sensor fusion do not propagate forward in the perception system world model. This makes these O-D sensors ideal for low cost perception of the robot's environment.

This work is laying out the concept for improved vegetation detection through the fusion of the IR and visual streams. We initially provide experimental results using a 360-degree far-IR camera coupled with three Kinect cameras. We also apply the techniques to an O-D IR camera and an O-D visual camera. The current sensors are mounted on top of a robot platform and we will develop a more robust data set and analysis. Fig. 2 is an example of the O-D IR, O-D Color, and Kinect cameras mounted on top of the robot, and a sample of a frame extracted from the scene by the IR and Color cameras.

The reason for choosing O-D IR and O-D vision is due to their superior performance in the following areas:

- Wide Field of View (FoV) - 360°
- Optical Flow or Structure from Motion – translation and rotational flow fields has different characteristic shape in spherical system.
- With camera and mirror axis perpendicular to the floor – vertical lines map to radial lines in the spherical coordinate system.
- O-D far-IR provides robustness in varying light conditions day or night
- A single O-D image completely defines the visual characteristics of a location.

Table 1. summarizes the camera setting.

The contribution of this work is to adopt an O-D far-IR camera and O-D Color camera on our mobile robot and developing a new fusion framework, called Thresholded Region Fusion (TRF) for semantic extraction multi-spectral fusion of O-D IR and O-D Color vision. Our approach will fuse a Modified Normalized Difference Vegetation Index (MNDVI), modified for far IR instead of near-IR (NIR) with a region based thermal semantic structure to improve the number of false detects with the traditional NDVI approach.

Table 1. Camera Settings

Sensor	Image Type	Advantage	Disadvantage
O-D IR	far-IR	Wide FoV	Low Resolution
Kinect	Red/Near-IR	High Resolution	Narrow FoV
O-D Color	Visual	Wide FoV	Mid Resolution

1.4. Introduction to Deep Learning Sensor Fusion

Our Deep Learning Sensor Fusion (DeepFuseNet) approach achieved the best overall results:

- Significant reduction in false positives from 32.5% for Modified Normalized Difference Vegetation Index to 2% for the DeepFuseNet.

While there has been a great deal of research in intelligent robotic perception, the practical use of Unmanned Ground Vehicles (UGV) to operate with small teams of humans in many civilian, service, and military applications, are limited by the current state of intelligent perception and the high cost of sensors. While the current state of obstacle detection and avoidance enables the handling of well-structured obstacles, it is limited in distinguishing between solid obstacles and low density passable objects. With improved utilization of deep learning and convolutional neural networks (CNN) coupled with omnidirectional (O-D) camera systems, the corrections required and errors introduced are reduced when creating a 360-degree fusion of our visual and IR O-D cameras. In addition, the computation requirements are reduced and sources of error are improved for the fusion of O-D infrared (IR) and O-D electro-optical cameras. Multi-spectral sources can also be fused with reduced error while the calculation errors for sensor fusion do not propagate forward in the perception system world model. This makes these O-D sensors ideal for low cost perception of the robot's environment.

The authors are leveraging the benefits of potentially low cost, O-D IR and O-D color vision sensors coupled with intelligent deep learning perception algorithms to the solution of this problem. This work lays out a concept for improved vegetation detection and the reduction of false detects through the fusion of the infrared and visual streams, and provides initial experimental results using our DeepFuseNet approach with an O-D far IR camera coupled with an O-D visual camera, and are compared to our previous work with region based fusion of O-D IR/Visual Kinect cameras and with the O-D IR/O-D visual camera are compared in the experiments section.

Fig. 3 is an example of the Pioneer robot with an O-D IR, and O-D visual camera, mounted on top, the Kinect cameras, and a blow-up sketch of the camera with internal mirror geometry overlaid are also shown. The reason for choosing O-D vision and IR is due to their superior performance in the following areas:

- Wide Field of View (FOV) - 360°
- Optical Flow or Structure from Motion – translation and rotational flow fields has different characteristic shape in spherical system.
- With camera and mirror axis perpendicular to the floor – vertical lines map to radial lines in the spherical coordinate system.
- O-D Far-Infrared provides robustness in varying light conditions day or night
- A single O-D image completely defines the visual characteristics of a location.

The contribution of this work is to utilize an O-D color camera on top of an O-D far IR camera by

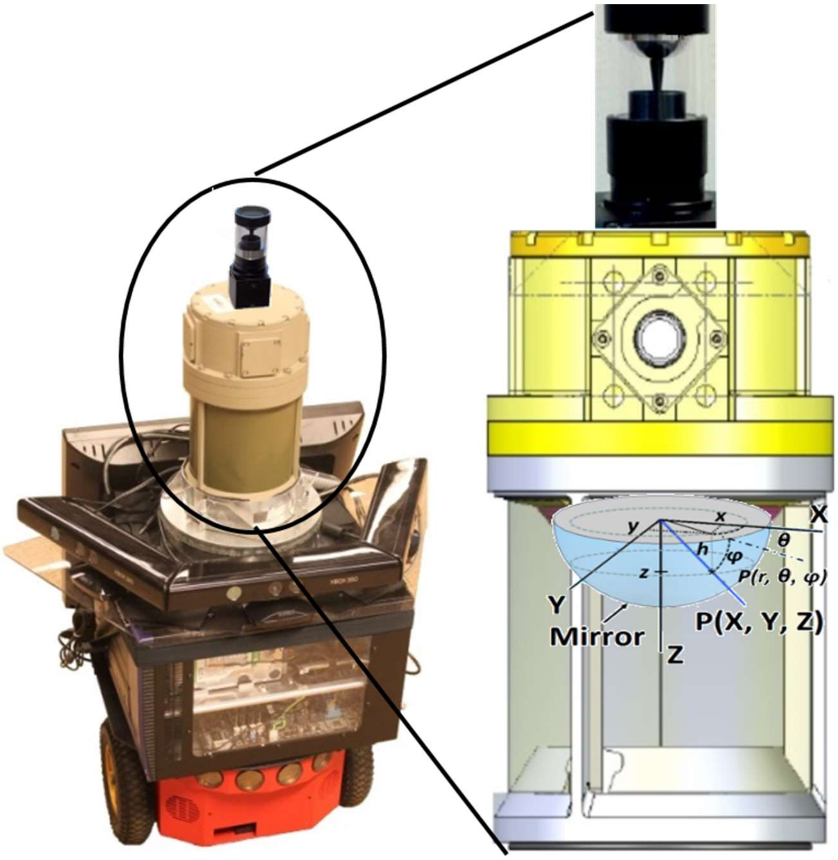


Figure 3. Robot with O-D IR, O-D Visual, and Kinnect Cameras

Robot with O-D IR and O-D visual cameras, and onboard computer. A blow-up view of camera with geometry overlay.

developing a new fusion framework, called a deep learning (DL) convolution neural network (CNN), or fusion network (DeepFuseNet) which provides for semantic scene extraction and multi-spectral sensor fusion of O-D IR and O-D vision sensors. Our approach will fuse the visual semantics with thermal region semantic structure to improve the number of false detects with the traditional NDVI approach.

1.5. Impact

The widespread interest in Unmanned Ground Vehicle (UGV) platforms for commercial, civilian, police, emergency response and military applications has increased the number of unmanned system applications throughout the world. The significance and utilization of these unmanned system applications has increased the emphasis and research in the area of intelligent perception and autonomous robot operation over the last several decades. The Office of Naval Research (ONR) conducted a Small Unit Mobility Enhancement Technologies (SUMET) study [1] with several significant findings:

- Across the Department of Defense (DOD) over $2B has been invested in autonomy.
- Current trend is focused on optimizing multi-modal sensors to compensate for poorly performing algorithms.
- Current state-of-the-art sensor suites are the major UGV cost driver and are cost prohibitive.

The Robotics Collaborative Technology Alliance (RCTA) identifies an R& D level 6.2 investment gap in three areas:

- Advanced Perception
- Intelligent Control Architectures and Tactical behaviors
- Human-Robot Interface

For Unmanned Ground Vehicles to become practical and cost effective in these applications, it is necessary to achieve a more robust and affordable perceptual framework. Finally, the application of O-D wide field of view sensors both visual and infrared will allow the agents to operate both in the day and in the night.

Goal Statement

The goal is to establish a new framework for the robot's scene semantics that can be recognized by the combination of a scene semantics feature extractor and artificial neural network fusion of the O-D infrared and visual stream approach. It is also hypothesized that backpropagation matching of the scene feature space and transfer learning from a larger data set can realize accurate 3D scene reconstruction with a semantic overlay. Therefore, we will seek three goals below:

Goal 1: Direct Spherical Calibration (DSC) of O-D far infrared (IR) camera

The proposed DSC will use a Direct Spherical Calibration (DSC) method to find the center of the camera, focal lengths, and camera properties of the calibration board images. Current O-D calibration methods are optimized for visual O-D cameras. The fuzzy edges in the thermal images make corner extraction more difficult, and the baseline calibration methodologies explored rely heavily on multiple corner selection from a calibration board grid. We improve the method by selecting the centroid from the four outside corners to use as the direct spherical input to a simplified re-projection calibration algorithm.

The algorithm will iterate to minimize the re-projection error of the spherical coordinates of the calibration board centroid while adjusting the intrinsic and extrinsic camera properties to match a simplified spherical calibration equation. To verify the robustness and improvement for estimation accuracy, Root Mean Square Error (RMSE) will be used as a criterion. The optimum calibration setting will be explored using a Pareto Optimization algorithm while minimizing circularity of the re-projected data, re-projection error, and computation time.

Goal 2: Vegetation feature extraction with an Auto-encoder Neural Network

For providing the vegetation detection using multi-spectral O-D far infrared and visual stream, a scene semantic feature extraction algorithm will be proposed. To build the semantic scene feature vector, we use a combination of a modified normalized vegetation difference index (MNVDI) spectral decomposition, and thermal region decomposition kernel filters as input into a self-organizing feed forward feature extractor implemented by a Sparse Autoencoder Artificial Neural

Network (AANN) algorithm, the performance of the proposed vegetation detection target modeling will be evaluated via infrared and visual data captured with our robot setting.

Goal 3: Robot semantic scene reconstruction using DeepFuseNet - Deep Learning Sensor Fusion (DLSF)

Based on the proposed DSC method in goal 1 we will apply transfer learning to a deep learning (DL) convolution neural network (CNN). We will use the proposed scene semantic feature extraction algorithm in goal 2, to extract the important features for recognizing the composition of the scene. We applied deep learning sensor fusion to semantic scene reconstruction using Deep Learning Sensor Fusion (DLSF) to reconstruct the scene semantics reconstruction modeling as input into transfer learning of a Convolution Kernel Neural Network (CNN) as a feed forward unsupervised scene semantic feature classification model. The scene semantics reconstruction will be implemented using the AANN Semantic Feature Extractor and the CNN Classifier mapped to the semantic scene geometry. The resulting semantic mapping will be matched with a semantic behavior vector for passibility decisions. Analyzing outcomes from the scene semantics reconstruction model will present the percentage of non-detects, false detects, and incorrect classification. The RMSE and statistical values will be calculated for each semantic feature to validate.

1.6. Innovations

The technical merits, science, and new contributions are comprised of the three major innovations given in Sections 1.6.1, 1.6.2, and 1.6.3 which is further described below.

1.6.1. Direct Spherical Calibration (DSC) of O-D far infrared camera

DSC investigates the calibration of O-D infrared (IR) camera for intelligent perception in unmanned system platforms. Current O-D camera calibration approaches are optimized for visual O-D cameras. A literature search found no significant research for calibration of O-D infrared cameras. The calibration of O-D IR cameras and the use of O-D IR vision have not been adequately addressed. With the IR image the edge boundaries are not as sharp as with color vision cameras. This leads to error in the identification of control points for re-projection, at the calibration board grid corners, which are extracted from the calibration board image and re-projected. The camera properties are iterated and the re-projection error is minimized in order to find the camera intrinsic and extrinsic properties In order to fully address O-D IR camera calibration, we propose a Direct Spherical Calibration (DSC) approach for a more robust method of calibration. The newly proposed method, DSC, will address this by using the centroid of the calibration board and its spherical coordinates to directly calculate the re-projection parameters. We compare DSC to three baseline visual calibration methodologies and augment them with output of the spherical results. We also look at the optimum number of calibration boards using an evolutionary algorithm and Pareto optimization to find the compromise between accuracy and methodology and number of calibration boards. The benefits of DSC are more accurate calibration board geometry selection, better accuracy, and less computation time than the two baseline visual calibration methodologies.

1.6.2. Vegetation Detection in 2D with Auto-encoder Neural Network

The fusion of O-D Infrared (IR) and vision sensors is proposed to increase the level of intelligent perception for unmanned system platforms. Current approaches are primarily focused on O-D color vision for localization, mapping, and tracking, and a literature search found no significant research in our area of interest. The combination of O-D IR and color vision for the extraction of feature material type, has not been adequately addressed. We will look at augmenting indices based spectral decomposition with IR region based thermal decomposition to address the number of false detects inherent in indices based spectral decomposition. Our work shows that the fusion of the Normalized Difference Vegetation Index modified for far infrared with the IR signature region representing vegetation minimizes the number of false detects seen with NDVI alone. The benefits of O-D IR region based thermal decomposition coupled with visual signature analysis for texture recognition and semantic extraction will be the contribution of this work.

1.6.3. Robot semantic scene reconstruction using DeepFuseNet

The final step will be the reconstruction of the scene with a semantic vegetation scene overlay through a neural network fusion of the O-D IR and vision streams. By fusing these proposed O-D IR and visual sensor streams utilizing DLSF and transfer learning techniques from networks trained on the ImageNet dataset, it is expected that reconstruction of the scene semantic structure will be realized. The significance, innovation, and technical merits of our three goals are shown in Table 2. More methodological details about these goals will be presented in the following sections. We present representative modules for the three goals in Fig. 4.

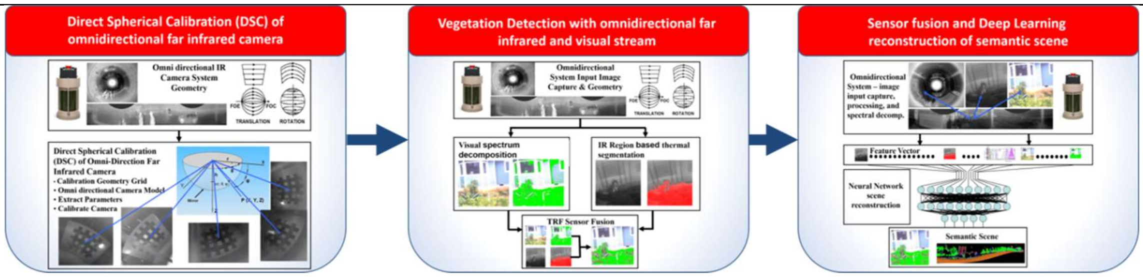

Direct Spherical Calibration (DSC) of omnidirectional far infrared camera
Omni directional IR Camera System Geometry
FOE
FOC
TRANSLATION
ROTATION
Direct Spherical Calibration (DSC) of Omni-Direction Far Infrared Camera
• Calibration Geometry Grid
• Omni directional Camera Model
• Extract Parameters
• Calibrate Camera
Mirror
P (X, Y, Z)
Vegetation Detection with omnidirectional far infrared and visual stream
Omnidirectional System Input Image Capture & Geometry
Visual spectrum decomposition
IR Region based thermal segmentation
TRF Sensor Fusion
Sensor fusion and Deep Learning reconstruction of semantic scene
Omnidirectional System – image input capture, processing, and spectral decomp.
Feature Vector
Neural Network scene reconstruction
Semantic Scene

Figure 4. Representative depiction of three goals

Direct spherical calibration of O-D IR camera, vegetation detection through index based and region segmentation fusion, and deep learning sensor fusion for vegetation detection.

Table 2. Significance, Innovation and Technical Merits

Step	Significance and Innovation	Technical Merit
1	• Simplify calibration board centroid capture • Improve the omnidirectional infrared camera calibration accuracy with the	• Provide a more robust, accurate, and lower computation time methodology for the calibration of infrared omnidirectional camera. • Provide a basis for 3D scene reconstruction in SA3

	Direct Spherical Calibration (DSC) algorithm.	
2	• Build the environmental vegetation feature extraction kernel from MNVDI based feature extraction. • Build the Infrared thermal region based feature extraction utilizing kernel feature filters. • Unsupervised feature learning using a 2D auto-encoder neural network.	• Provide a fusion technique, which overcomes the false detect problem in the semantic overlay in the reconstructed images. • Provide a robust unsupervised feature learning tool for 2D material feature extraction. • Provide a 2D semantic map of the scene to better identify object materials in the scene and identify passable obstacles. This will better achieve semantic scene mapping in SA3.
3	• Apply unsupervised machine learning to extract 3D feature map • Extend the Autoencoder and add artificial Convolution Kernel Neural Network (CKNN) semantic model	• Develop the artificial convolution kernel neural network sensor fusion model based on the infrared and visual streams from the scene images and apply a 3D semantic scene overlay of the materials in the scene to aid improved robot perception. • Improve robot perception to allow the robot agent to better understand the semantic construct of its environment, and enhance the accuracy of semantic mapping, path planning, and scene reconstruction

1.7. Summary

The innovation of this work is the improvement in robotic vegetation detection utilizing a deep sensor fusion network architecture which reduces false positives in vegetation detection while maintaining a reasonable recognition rate. The recognition rate achieved is 95.6% with a 92% reduction in false positives over the classical index based approach.

The remainder of this Thesis is organized as follows: Chapter 2 is prior related works. Chapter

3 is the Technical Approach for the three methods. Chapter 4 is the Experimental Results for the three methods. Chapter 5 is the Conclusion for the three methods.

The best performance was achieved by the DeepFuseNet approach with 95.6% recognition and 2% false positives. Fig. 5 shows the training and deployment concept and is further explained in Chapter 3 Section 3.3.6.

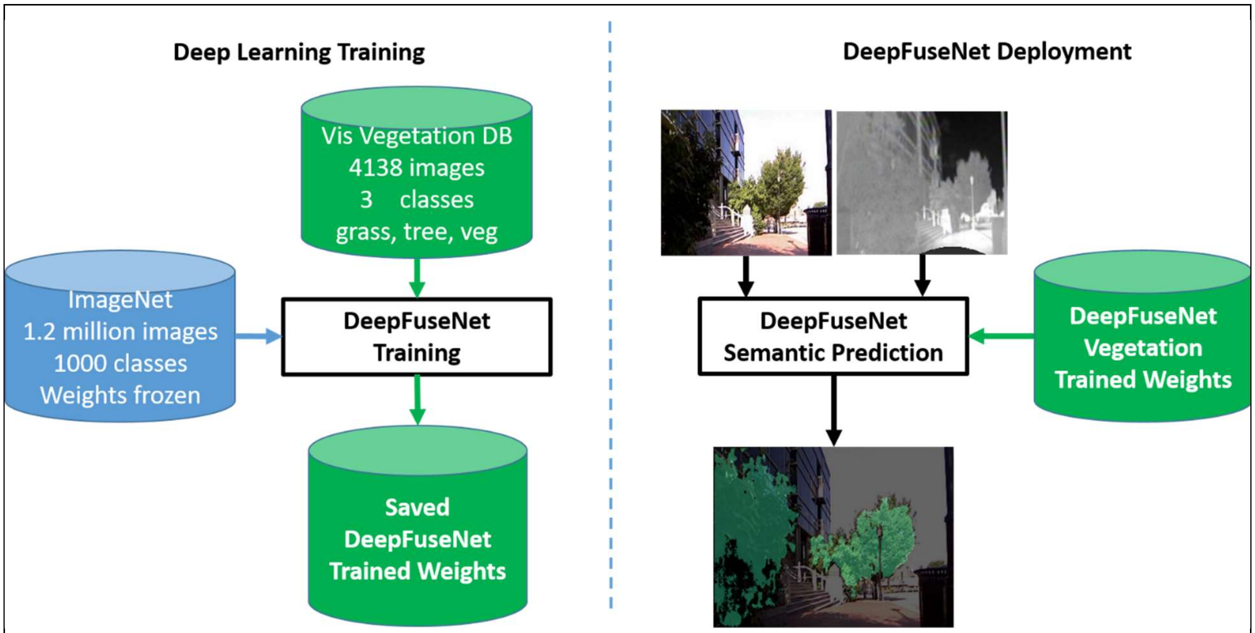


Figure 5. Concept level depiction of Deep Learning Training and DeepFuseNet deployment

On the left is the high level Deep Learning training pipeline, on the right is how the DeepFuseNet is deployed. The DeepFuseNet Training block is detailed in figure 27 in Chapter 4 Section

2. Chapter – Prior Related Works

Chapter 2 covers related works and is organized as follows: Section 2.1 covers O-D Camera Calibration, Section 3.2 describes some prior work in O-D applications, and sensor fusion, Section 4.2 describes some prior work in O-D applications, and sensor fusion,

2.1. Prior Related Works for Omni-direction Camera Calibration

Section 2.1 covers O-D Camera Calibration and is organized as follows: Section 2.1.1, Omnidirectional Camera Geometry, Section 2.1.2., Omnidirectional Visual Camera Calibration Methodologies, Section 2.1.3., and Omnidirectional IR Camera Calibration for Geometry Transformation, Section 2.1.4 Non-standard methods, Section 2.1.5., Omnidirectional IR camera calibration methodology Pareto optimization.

2.1.1. Omni-direction camera geometry

There has been a large body of research in the area of omnidirectional camera geometry, and calibration of omnidirectional visual cameras. There is little work in the calibration of IR omnidirectional cameras. The various methodologies are summarized in Table 3.

Geometrical Approaches - 3D omnidirectional geometry approaches are presented in [1-4]. The central catadioptric imaging process was shown in [5] to be a two-step projection process; first from the 3D world point $\boldsymbol{X} = (x, y, z)^T$ to a point on the unit sphere $\boldsymbol{X_s} = (x_s, y_s, z_s)^T$, and from the sphere to a point m on the image plane $\boldsymbol{m(u, v)}$.

Table 3. Methodology Summary

Method	Ref,	Model	Approach	Views
Geometry	[1-5]	N/A	Mixed	N/A
Survey	[6], [18]	N/A	Mixed	N/A
DLT	[11]	Sphere	Linear Eq.	Single
Distortion 2D	[7-8], [12],[13]	Distort	Polynomial	Multiple
Two-step Estimation	[10]	Planer Motion	Joint Optimization	Single & Odometric
Overlapping views	[14]	Lucas-Kanade	Bundle adjust	multiple
3D Reconstruct	[15]	Multiple Observation	Mixed	Multiple
Spherical 2D	[10],[18]	Sphere	Single viewpoint	Multiple
Spherical Lines	[3-5, 16] [19, 23, 28]	Sphere	Line projection	Single
Straight lines, line scan, lines from image	[20-21], [31]	Lines from motion, photogrammetry, lines from LIDAR and image	Vanishing viewpoint, Multi-camera	multiple
Visual motion	[22]	Camera Pose - two spherical images	Bayesian Uncert. Analysis	Moving camera
Moving objects	[30]	N-view matching	SFM	multiple
Generic	[33, 34]	Planes	Multiple	Single
Multiple Spherical Surfaces	[30]	2 mirror optics	IR point sources	Single
Infrared	[35-37]	Not described	Not described	various
Non-standard	[38-39]	Optical flow depth map	Multi axis stereo	multiple
Pareto	[40-41]	Pareto Frontier	Pareto Optimize	N/A

2.1.2. Omnidirectional visual camera calibration methodologies

Omnidirectional cameras are being applied in computer vision and robotics because of their advantages in wide field of view (FOV). Much work has been done on calibration methods optimized in the visual band but very little in the IR band.

Survey of Methods - [6] of about 25 methods was categorized into four principal approaches.

- ***Direct Linear Transform (DLT) approach*** – uses a spherical model and obtains a closed form solution using 3D – 2D correspondences.
- ***Spherical-2D Pattern*** **–** uses a spherical model coupled with multiple views of 2D pattern with as many points as possible in the pattern.
- ***Distortion 2D Pattern*** **–** Models the 2D images as distorted images and uses a Taylor expansion polynomial approximation as the projection function to find the distortion parameters.
- ***Spherical Lines*** – This approach uses the spherical camera model, a single omnidirectional image with at least three lines. The 3D lines are mapped to conics in the omnidirectional image.

[6] evaluated the four primary methods of omnidirectional calibration. Three of these methods (DLT-like, Spherical 2D Pattern, and Distortion-2D Pattern) showed good results with very similar performance. Of these three, the Distortion-2D Pattern approach had the best results in real world applications. All of these methods have been applied to omnidirectional color vision cameras but not to IR omnidirectional cameras. Three of the four primary methods identified in [6] were based on spherical geometry.

The first two Baseline methods we used for comparison were chosen based on the best two methods found for visual omnidirectional camera calibration in the author’s literature search. We also added a third more generic method for comparison. The three methods chosen to compare were Baseline 1: The Distortion-2D Pattern approach used by [7-9], and the Baseline 2: Spherical-2D Pattern approach [10]. The review of a radically different approach was also evaluated, Baseline 3: a generic calibration model [35, 36].

Other methods - were briefly evaluated and not used. A Direct Linear Transform (DLT) like approach is presented in [11].

The presentation of a hand-eye camera calibration approach which takes into account a lens distortion camera mode is given in [12-13]. The approach optimizes the intrinsic and extrinsic camera parameters using an iterative extended Kalman filter.

A 360 degree Distributed Aperture System composed of 6 narrow field of view cameras [14] is presented and uses forward additive Lucas-Kanade algorithm with bundle adjustment strategy to solve for offline estimation of camera orientation.

A multiple observation 3D reconstruction is presented in [15]. A minimization of the re-projection error assuming that the angle of incidence between the point on the mirror is equal to the reflection angle or angle to the center of the mirror [16].

[17] also did a survey of omnidirectional calibration methods and characterized the methods by geometrical approach such as 3D to 2D, straight line, spherical and mixed approaches. Calibrating the para-catadioptric camera from the projection of a sphere onto the image plane is also presented in [18].

[19, 20] use the fact that lines in space map to circles in an omnidirectional image leveraged to find points on the line projections, and then apply best fitting circles to those points. The authors use an offline calibration of surveillance cameras using camera height coupled with line geometries extracted from moving vehicles in the scene [21]. The authors in [22] use a moving fisheye camera to study camera pose uncertainty in a pipe inspection setup. Calibration was accomplished using

the [8] omnidirectional toolbox.

The relationship of line and sphere invariant geometry and the projection of lines to conic curves are further investigated [23, 27, 28]. The authors present a self-calibration method using projection rays to individual pixel matches between images [24-25]. An approach for nonlinear localization of corner points to least squares fit the corner gray scale model to the image is presented [26]. An omnidirectional sensor is used in [29] to map the 360-degree environment around the robot. [30] Calibrates a multi-axial imaging systems consisting of a camera viewing multiple spherical reflecting or refracting surfaces to achieve wide angle views. Then from the rays of two or more spheres the pose of a single calibration grid is obtained by linear decomposition independent of the sphere locations and radius. In [31], the authors use the environment to find points, lines and planes in both the image and the laser data to identify trihedron of lines from which to calibrate the camera and the laser. The four outside corners are selected and then the internal control points are extrapolated. The four outside corners are selected and then the internal control points are extrapolated. [32] is the MATLAB calibration toolbox. The authors [33-34] give a generic calibration method that we used as our third baseline comparison.

2.1.3. Omnidirectional IR camera calibration for geometry transformation

Very little work has been done in the area of IR omnidirectional sensors or sensor calibration. This is the reason our work is important. An evaluation of five IR reflective hot mirror materials and a log-polar mapping technique is presented in [35], but they do not present any calibration information. A box with four thermal point sources is used to calibrate the camera, but do not describe their calibration methodology [36]. The authors in [37] use thermal cameras to measure odometry on a robotic platform. The calibration is accomplished using the standard checkerboard approach with the application of aluminum grid to improve the thermal reflectance.

2.1.4. Non-standard methods

In [38], the authors use an array of multiple micro lenses focused on the various scene light fields from the single primary lens to extract depth information from the different light fields from the scene. As a result, the authors take advantage of the geometry to perform a non-parametric calibration based on optical flow. The authors in [39] use the optical properties of water-drops on glass to estimate a depth map using stereo from multiple water drops. The method has limitations because the perspective camera cannot get high resolution images through the water drops.

2.1.5. Omnidirectional IR camera calibration methodology Pareto optimization

Pareto optimization is a methodology commonly used for multi-variant optimization problems. Visualization techniques are described for multi-dimensional optimization problems using multivariate mesh displays, color mappings, and multiple views [40]. Techniques for multi-variant objective optimization are given in [41].

2.2. Prior Related Works for Index and Vegetation Region Segmentation Methods

Robust and inexpensive intelligent perception is a key enabler to the practical application of Unmanned Ground Vehicles (UGV) operating with teams of humans in areas of police, rescue, and military applications. In addition, truly commercial applications will benefit from this research and other intelligent perception work as well. Our review of the literature highlights that the focus of recent research with O-D cameras has been primarily in the area of improvements in mapping, localization and tracking, robot navigation, and obstacle detection.

The authors believe that the fusion of O-D IR and Vision has the potential to provide 3.2.1 *Omni-direction Camera Setting*, Section 3.2.2 *Visual and IR Index Based Vegetation Detection*; Section 3.2.3 *IR Stream Segmentation using Region based Thermal Analysis*, and Section 3.2.4

Sensor Fusion Methodologies.

2.2.1. Omni-direction Camera Setting

The spherical geometry characteristics of the O-D sensor are leveraged. Table 4. summarizes the literature review for this section.

The O-D geometry approach exploits the radial straight line geometric feature of O-D vision to map to semantic primitives of the environment (doors, buildings, walls, edges, trees, corners, and radiators) and track these primitives as the robot moves through the environment. There are several approaches to localization and homography (determining the visual geometry of the scene) [42-47]. [42] applies O-D vision and odometry to self-localization of a UGV in a non-static environment, and to determine robot's global pose (x, y, θ). They also looked at several sensor fusion models and whether fusing data then detecting/classifying (low level fusion) or the utilization of individual sensor modality to detect/classify and then fusing the results (high level fusion) gave better overall results. Our approach in this chapter is to use feature level fusion coupled with segmentation techniques as described in section 3.3.

Table 4. Omnidirectional (O-D) Camera Applications Summary

Method	Reference	Model	Approach
Localization	[42]	SLAM	Mixed
Homography	[43, 44]	Visual Geometry	Mixed
Navigation	[45]	Sphere	Optical Flow
Calibration	[46]	O-D camera calibration	Direct Spherical Calibration
Robotic Feature Tracking	[47]	Vertical Line Recognition	Robust Feature Descriptor

Homography (Visual Geometry) estimation from feature points extracted from a moving Omni-vision sensor is another popular approach used in both [43], and [44]; [43] uses multiple homographies from virtual image planes in O-D vision pairs. Navigation is explored in [45 The

Table 5. Vegetation Detection Methodology Summary

Method	Reference	Model	Approach
NIR	[48,49]	Index Based	Satellite Remote Sensing
Multi Spectral	[50]	Hyperspectral	Mixed
NDVI	[51 - 53]	Various	Improve NDVI
NDVI Variant	[54]	Principal Component Analysis	Statistical Framework
NDVI	[55-56]	Topological Effects	Sensitivity Analysis

authors calibration approach for our O-D camera is presented in [46]. Identification of vertical line geometry in the image is addressed in [47].

2.2.2. Visual and IR Index Based Vegetation Detection

Vegetation detection is required because the identification of vegetation aids the robot in detecting materials that it can pass through such as grass or small bushes. Table 5. summarizes the literature review for this section. [48-56] address different aspects of vegetation detection. However, these approaches are prone to false detects which we hope to minimize through our approach.

The identification of vegetation is especially useful. If the object detected is made out of leafy vegetation or grass, it is more likely passable than a tree trunk or a man-made object. The use of the NIR spectrum in addition to the visual spectrum has been shown to provide useful information in the detection of vegetation in remote satellite imaging [48, 49].

The application of multispectral and hyperspectral techniques is explored in [50]. [51-56] Normalized Difference Vegetation Index (NDVI) is a very popular index approach to vegetation classification in remote sensing. However, these approaches that are applied in satellite imagery are taking a macro look at large data sets to classify ground cover. In the NDVI approach a comparison of the levels of red reflectance to the NIR reflectance has been shown to correspond to vegetation.

2.2.3.IR Stream Segmentation using Region based Thermal Analysis

Segmentation and fusion techniques have been explored [57-66], and both region based and edge based approaches have been demonstrated. [57] augments the NDVI with 3D LIDAR point cloud to compensate for the vegetation index shortcomings; this is an approach which utilizes expensive sensors. Table 6. summarizes the literature review for this section.

[58] evaluates the calibration of Spectral Response Functions from multiple sensors to improve the fusion of Multi-sensor NDVI time series. Our approach will fuse low cost optical and IR sensors to minimize the false detects. Localization from O-D vision using RBG color histograms [59-61] to match images in the database of regions the robot may visit is another approach.

Region Segmentation techniques have been explored [62-26], and both region based and edge based approaches have been demonstrated. [62] apply a multi-mode sensor fusion of Lidar, color vision and near-IR cameras fused to provide perception of the terrain around a mobile robot.

Localization from O-D vision using RBG color histograms [63], and a similar approach saves computing power by extracting a numerical signature gray scale histogram using Haar Discrete Wavelet Transform [64]. This approach also uses expensive Lidar sensors which we are trying to avoid. Histogram approaches have also been used for region segmentation [63-66] using clustering techniques based on histogram thresholds.

Table 6. Segmentation and Histogram Method Summary

Method	Reference	Model	Approach
LIDAR and NDVI Fusion	[56]	Sensor Fusion	Fusion of Lidar and NDVI results
Multi-sensor	[57]	NDVI time series	Spectral Response Functions
Histograms	[58-60]	RGB and Gray Scale Histogram	Region matching
Region Segment	[61-65]	Mixed	Region and Edge, Histograms

Table 7. Sensor Fusion Method Summary

Method	Reference	Model	Approach
Structured overview	[67]	Survey	Various
Extended Kalman Filter	[67]	Decision fusion	Composite process health index
Raw Data Level Fusion	[68-69], [74], [80], [84]	Data fusion	Improve signal to noise similar sense
Feature Level Fusion	[70-71], [78-79], [81-83]	Feature fusion	Feature Extraction at sensor level
Decision Level Fusion	[72-73], [76-77]	High level decision	Individual sensor analysis high level probabilistic confidence fusion
Human-Robot Fusion	[75]	Bayesian Soft Max	Gaussian priors and human guidance
IR-Vision Fusion	[85]	Visual Gradient and infrared intensity	Feature Level Fusion
IR-Vision Fusion	[86]	Non subsampled Contour-let Transform	Fuse sub-bands and low level features
IR-Vision Fusion	[87]		
Label-Me	[88]	Label-Me	MIT CSAIL Labeling Tool
Compressive Data Fusion	[89]	Compressive Fusion	Compressive Random Projections
NDVI	[90-91] [92-93]	NDVI	Analysis of ground cover for loss of forest cover.
Lidar-Hyperspectral Fusion	[94]	Lidar point cloud plus Mixture-tuned matched filter for Hyperspectral	Lidar – canopy structure, Hyperspectral – species spectral signature

2.2.4. Sensor Fusion Methodologies

Sensor Fusion techniques have been explored in several multi-sensor approaches [67-91]. Table 7. summarizes the literature review for this section.

The following are presented:

- Local Point Statistic
- Conditional Local Point Statistic
- 2D-3D Feature Fusion
- Laser Remission
- NDVI-Laser Fusion
- SVM Multi-Cam Fusion
- Multi-Cam NDVI Fusion

[67] provides a survey of sensor fusion methods which are categorized into three levels:

1) Raw data level fusion – sensors with similar characteristics are fused to improve signal to noise [68-69], [74], [80], [84]
2) Feature level fusion – features are extracted from different types of sensors and then similar features are fused for improved confidence [70-71], [78, 79], [81-83].
3) Decision level fusion – the sensors are processed individually and then the relevant information is then fed into a separate decision processor to decide on the most confident outcome [72-73], [76-77].

Where [72] applies decision level fusion to fuse the process sensors of vibration, noise, and force. [73] applies sensor fusion to robot off-road navigation, fusing individual sensor results by probability of blocked / unblocked and feeding this into high level behavior decision model. A data level fusion of sensor signals [74] is used to develop a thresholded composite failure index that combines both data fusion and degradation modeling to establish the composite index. The fusion of multiple machining quality sensors. [75] characterizes forest canopy using airborne LIDAR and hyperspectral feature based fusion. [76] applies data fusion to an omnidirectional and PTZ camera for optimal multi target tracking.

The fusion of environmental features [77] from visual and IR cameras is used to calculate the probability of pedestrian location. The automatic scene segmentation into salient features is accomplished by different methods and their results fused. The fusion of depth and color features [78] is used to distinguish crops from weeds. Feature fusion of visual and Synthetic Aperture Radar (SAR) is used to extract roads and buildings [79].

[80] uses the data level fusion of IR and Visual camera, preserving the intensity distribution of the IR image and the gradient variation of the visible image. [81] surveys several feature fusion methods including; substitution techniques-based, independent component analysis (ICA)-Based,

principal component analysis (PCA)-based, segmentation-based, Neural Network-based, mathematical morphology-based, and Multi Scale Transform (MST)-based fusion schemes. [82-83] utilize feature extraction at the sensor level. [84] provides fusion at the raw data level. [85] applies Gradient Transfer Fusion to extract the gradient information from the visual image fused with minimization of the total variation of the infrared intensity data. [86] applies a Multi-Scale Transform using a Non-Subsampled Contour-let Transform to extract high and low frequency sub images and fuse with low level features. [87] uses region growing for segmentation. The authors applied the MIT CSAIL LabelMe database and web tool for image annotation to label the images [88].

2.3. Prior Related Works for Deep Learning Sensor Fusion

Robust and inexpensive intelligent perception is a key enabler to the practical use of Unmanned Ground Vehicles operating with teams of humans in areas of police, rescue, and military applications. Commercial applications will also find benefit from this research as well. The authors believe that the fusion of O-D IR and O-D Vision has the potential to provide improvements in intelligent robotic perception. This section covers related works and is organized Section 4.2.1 Visual and Infrared Index, Histogram, and Region Segmentation Based Vegetation Detection, segmentation, Section 4.2.2 Deep Learning - Convolution Neural Networks (CNN), and deep learning to scene recognition.

2.3.1. Visual and Infrared Index, Histogram, and Region Segmentation Based Vegetation Detection

Vegetation detection is required because the identification of vegetation aids the robot in detecting materials that it can pass through such as grass or small bushes. Table 8 summarizes the literature review for this section.

The identification of vegetation is especially useful. If the object detected consists of leafy vegetation, it is more likely passable than a tree trunk or a man-made object. The use of the near infrared (NIR) spectrum in addition to the visual spectrum has been shown to provide useful information in the detection of vegetation in remote satellite imaging. [95], [96], the application of multispectral and hyperspectral techniques is explored in [97], [98]. However, these approaches that are applied in satellite imagery are taking a macro look at large data sets to classify ground cover. Bradley et al. [99] explore the application of the Normalized Difference Vegetation Index (NDVI) to vegetation perception in the DARPA Preceptor off road UGV. A comparison of the levels of red reflectance to the NIR reflectance has been shown to correspond to vegetation.

Table 8. Vegetation Detection Methodology Summary

Method	Reference	Model	Approach
NIR	[95-96]	Index Based	Satellite Remote Sensing
Multi spectral	[97]	Hyperspectral	Mixed
NDVI	[98-99]	Topological Effects	Sensitivity Analysis

Table 9 summarizes the histogram and segmentation approaches. Histogram and region segmentation approaches have also been applied to vegetation detection [100-108]. [100] applies the fusion of visual and LIDAR point clouds. [101-104] apply histogram approaches, and [105-108] apply region segmentation approaches.

2.3.2. Deep Learning Convolution Neural Network Fusion of O-D IR and O-D Visual Stream

Table 10 summarizes the literature review for this section. Neural Network learning is applied to obstacle avoidance in [109-138].

Table 9. Histogram and Segmentation Method Summary

Method	Reference	Model	Approach
LIDAR & NDVI Fusion	[99]	Sensor Fusion	Fusion of Lidar and NDVI results
Histograms	[100-104]	RGB & Gray Scale Histogram	Region matching
Region Segment	[105-108]	Mixed	Region and Edge, Histograms

A three-layer Neural Network (NN) is trained to recognize red colored visual objects [109] and the output of the NN is three movement commands (Forward, Turn Right, or Turn Left). Mapping of the objects in the robot's environment [110] is divided into two map representations, a set of perceptual maps from each sensor derived by a Neural Network Feature Extractor and a Self-

Table 10. Neural Network and Deep Learning Method Summary

Method	Reference	Model	Approach
Neural Networks	[109-111]	Supervised Learning	Terrain Classification
Neural Networks	[112-115]	Unsupervised learning	Face Recognition
Autoencoder	[116-119]	Input Matching	Feature Extraction
Deep Fusion Network	[120]	Residual Network	
Classifier Fusion	[121]	Low and High Level Classifier	Classifier Fusion
CNN	[122-128]	Mixed	Road signs, road boundary
Databases	[129-130]	ImageNet, City Scapes	Classification databases
Deep Learning	[131-138]	Various	Transfer learning, finetuning

Organizing Map algorithm, and a spatial map representing the location of each of the objects in the perceptual maps which is built up by a Growing Cell Structure NN approach.

Dynamic obstacle motion is tracked using a multilayer feedforward Artificial Neural Network (ANN) [111] by fusing ultrasonic and visual cues. The ANN is trained off-line using a relative error backpropagation algorithm. On-line the ANN predicts in real time the distance to the dynamic and stationary obstacles. [111] Applies an environmental predictor ANN rather than a motion predictor to provide the robot with what areas will be occupied by obstacles. The algorithm fuses data from multiple ultrasonic sensors to identify the correlation between them and predict the future sensor reading.

The use of unsupervised learning for classification of terrain travers ability is explored in [112]. On-line learning is achieved by establishing a correspondence between the vehicles navigation experience (successful traversals, slippage, collisions) using onboard sensors (IMU, bumper, and motor current), and visual characteristics of the terrain from a stereo vision sensor. The travers ability level is established as a travers ability affordance or rating of terrain difficulty to be traversed.

Unsupervised feature learning [113-114], is applied in [115] to face recognition. The method used applies two simple learning algorithms 1) a K-Means clustering algorithm to pre-filter the data, and 2) agglomerative clustering to group simple cells into "complex cells" that are invariant features. An analysis of the application of single-layer networks to unsupervised learning is given in [115]. In this work they apply several unsupervised learning algorithms followed by a convolution (conv) classifier to evaluate the effect of different parameters used in unsupervised learning and CNN.

Autoencoders [116-119] which are a class of Neural Networks where the network is trying to approximate the identity matrix are used as feature extractors where the learned weights of the sparse hidden layer represent features in the image. Rather than use labeled data, the Autoencoder tries to match the output to the input and thus learn feature patterns in the data.

Converting high dimensional data to low dimensional data using Autoencoders [116], and a method of initializing weights that enable a deep Autoencoder network to operate more efficiently is presented. The method is applied to character recognition and facial feature recognition.

Intentional noise corruption of random pixels is introduced and a DE noising Autoencoder [117] is applied to a deep learning network to provide a more robust initialization of the network by guiding the intermediate steps based on the correction of the corruptions. While [118] looks at efficient sparse Autoencoder techniques. [119] presents a two-stream architecture the first being a de-noising autoencoder to encode pixel spectral values and the second stream is the image being evaluated. These are then fused using a CNN.

[120] applies residual learning to optimize CNN layer learning, and fuse the output of different hierarchical layers of the network. [121] Evaluates several low and high classifiers and looks at the fusion of the classification results for color, texture, and geometry for terrain classification. [122] applied automatically generated training data for CNN building detection.

[32-37] apply CNN to the classification of terrain and vegetation in the environment.

In our work we utilized the ImageNet [129] and City Scape [130] databases. We also referenced [131-134] for refining our deep learning approach. [135] was our prior paper that we used for comparison to our prior methods. [136] applied automatically generated training data for CNN building detection. [137] applied sparsity for unsupervised deep feature learning. [138] applied deep learning to building extraction in remote data. The authors in [138] applied data augmentation to expand their training set.

3. Chapter – Technical Approach for the Three Methods

3.1. Technical Approach for Direct Spherical Calibration of Omnidirectional Far Infrared Camera System

Chapter 3 proposes our new work Direct Spherical Calibration (DSC), starting Section 3.1.1, Omnidirectional IR camera geometry, Section 3.1.2, Direct spherical omnidirectional IR camera calibration methodology, and Section 3.1.3, Optimal number selection of image capture of calibration boards, and Section 3.1.4 Comparison to other methods. describes some prior work in omnidirectional camera calibration.

3.1.1. Omnidirectional IR camera geometry

Using parabolic reflecting surfaces, a projected 360-degree field of view is created and transformed into a single 360 linear plane. With a single omnidirectional vision approach, Fig. 6, reconstructions of scenes create a reduced set of unknowns in the equations as compared to the many camera approach. The camera, parabolic mirror, and the geometry of the focusing lens and mirror are shown in Fig. 6, and is described in the equations (1-4) below:

$$r^2 = {x_p}^2 + {y_p}^2 \quad (1), \qquad \theta = \tan^{-1}\left({}^{y_p}\!/_{x_p}\right) \quad (2)$$

$$\varphi = \cos^{-1}({}^{z}\!/_{r}) \quad (3), \qquad z = {}^{\left({x_p}^2 + {y_p}^2 - h^2\right)}\!\Big/_{2h} \quad (4)$$

The IR camera is mounted in the housing above the parabolic lens, with a hole in the center of the lens. The point P in space is reflected from the parabolic mirror down to the base mirror P' and focused by the lower lens through the hole in the center of the parabolic mirror to the camera lens and image plane. F is the focus of the parabolic mirror.

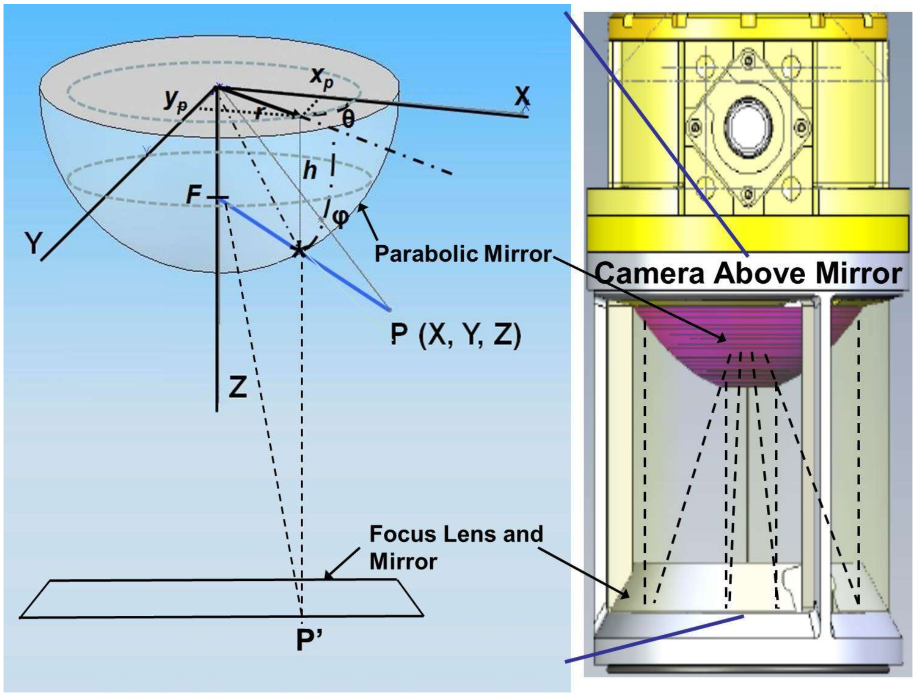


Figure 6. Omnidirectional Camera Geometry
shows the layout of the camera and the geometry of the omnidirectional parabolic mirror, and the relationship of the camera and focusing mirror/lens in the bottom of the structure. The Omnidirectional Camera geometry is presented, showing the relationship of the scene point P to the spherical point ρ and the x, y, z rectangular coordinates to the r, θ, φ spherical coordinates.

Where x_p and y_p are the pixel In the coordinates of the pixel, and P is the point in space being observed by the omnidirectional camera. spherical coordinate system θ and φ define the direction of the ray and r is the distance to the origin of the mirror in pixel coordinates and θ ranges from 0° to 360° around the edge of the mirror, and the pitch φ, ranges from 0° when pointing straight down, to 90° when pointing at the horizon. Fig. 6 shows the coordinate system in the O-D camera case, relating the (x, y, z) point to the sensor coordinate in (r, θ, φ). The camera geometry is shown in Fig. 2 Section 2.3.4.

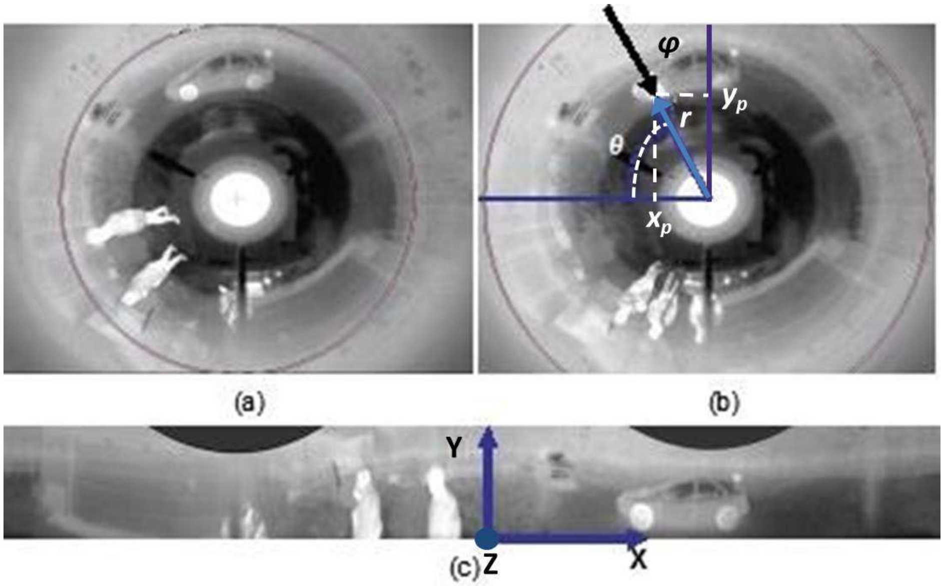


Figure 7. Omnidirectional Geometry Relationship
(a) Shows the omnidirectional image, and (b) Shows the O-D IR image with the r, θ geometry overlaid. (c) Shows the unwrapped rectangular image, and the X, Y, Z geometry axis.

The (x, y) points in the world coordinates are related to (x_p, y_p) by equations (1-4). The orientation, Fig. 7 a) shows the omnidirectional IR sensor. The sensor annotated with the two coordinate systems is shown in Fig. 7b) with the r, θ, and φ parameters overlaid. Fig. 7c) is the unwrapped image showing the relationship to x, y, and z coordinates, r is related to x, y by (1), and theta (θ) and Phi (φ) to x, y, by (2) and (3), z is related to r and h by (4).

3.1.2. DSC omnidirectional IR camera calibration methodology

With the IR image the edge boundaries are not as sharp as with color vision cameras. As a result, the corner selection algorithms and automatic methods do not accurately find the corners. This affects both the error and the scatter in the error results. The newly proposed method, DSC, will address this by directly using the spherical coordinates of the centroid of the calibration board, found from the four outside corners of the calibration board grid, and will use the spherical coordinates of the calibration board center to directly calculate the re-projection parameters.

Fig. 8 shows a block diagram of the geometry transformation and the relationship of the spherical mirror to the captured calibration boards.

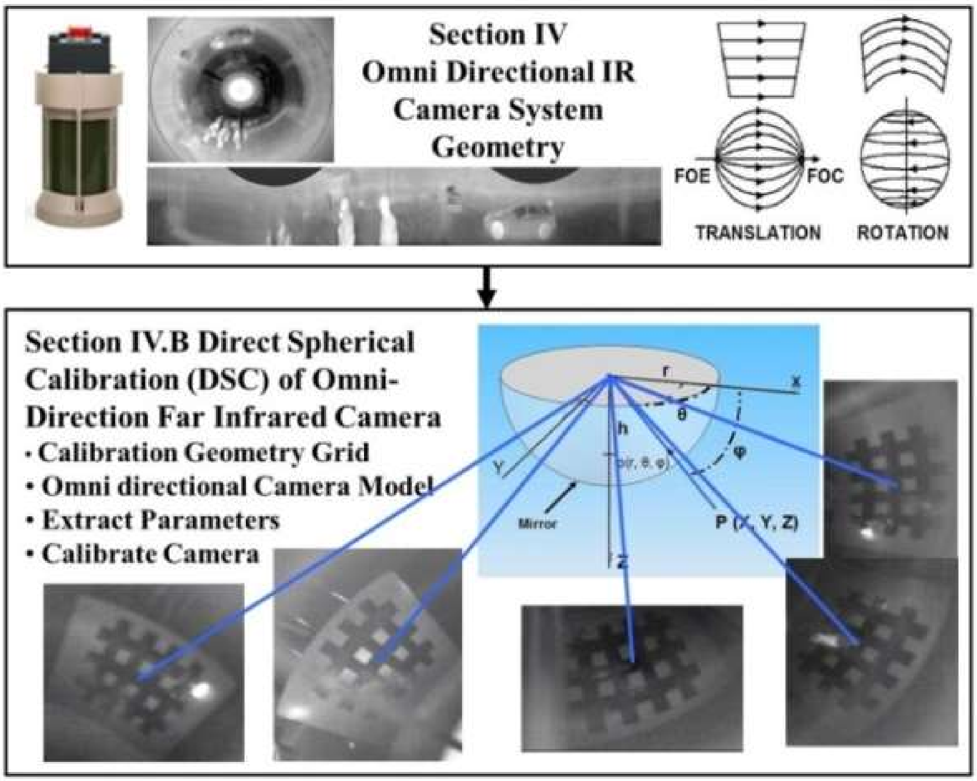


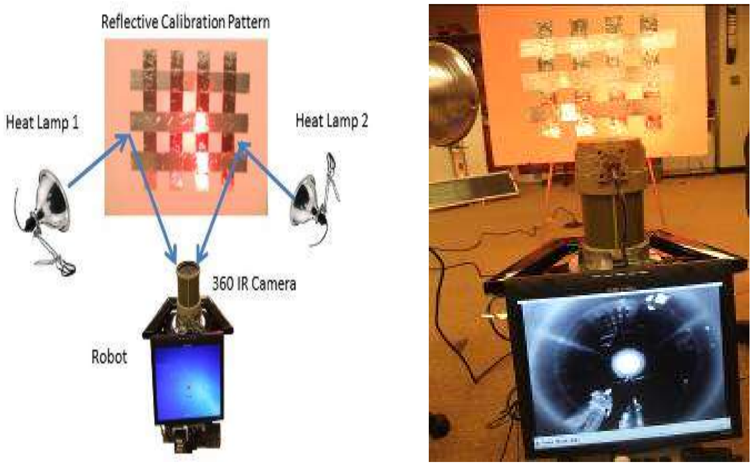
Figure 8. Camera Calibration Overview
Capture the location of center of calibration boards at different radius and angle from the omnidirectional IR camera.

The calibration process uses the planar calibration pattern to capture a matrix of spherical coordinates of the calibration board center points and then minimizes a least squares minimization function of the difference between the actual and re-projected points to converge on the camera intrinsic and extrinsic calibration parameters.

The proposed DSC consists of the six steps as follows.

STEP 1: ESTABLISH A THERMAL CALIBRATION GRID 2D PLANAR PATTERN.

The calibration setup is shown in Fig. 9 and consisted of a pattern of reflective tape applied on a closely laid out grid with +/- 0.03215 accuracy on a white board with two IR heat lamps shining on the surface. The calibration board was calibrated by careful measurement of the grid spacing. Fig. 8a). Fig. 8b) shows the sensor on the robot with the board and heat lamps repositioned around the robot and camera. Three data sets were captured, sparse consisting of ten (10) images, moderate consisting of twenty-five (25) images, and dense consisting of forty-seven (47) images. The three setups were done on different days with the board and lamps being repositioned for each data image. The reflected image was then captured by the omnidirectional IR camera shown on top of the robot in Fig. 8a and 8b.



a) Camera Calibration setup

b) Calibration setup

Figure 9. Camera Calibration setup:
a) Calibration setup diagram, with two heat lamps and a calibration board with thermally reflective pattern b) the calibration pattern illuminated by)IR lamp with robot in foreground displaying Omnidirectional IR image.

These images were then processed by the three algorithms for comparison. The method used in [31] to capture a calibration pattern from an LCD screen would not work with our IR O-D camera, due to the constant temperature of the screen. The methods used in [22 and 37] both used multiple cameras and were more complex than our setup. [22] Required a frame camera and a line scan camera, while [37] required an LCD projector, two stereo cameras and an IR camera. [30]

used a single camera looking through multiple spherical lenses on different axis, this again was more complex than our setup and did not apply to our camera setup.

STEP 2: DIRECT SPHERICAL CALIBRATION (DSC) CAMERA MODEL.

The DSC methodology uses the direct spherical coordinates of the calibration grids in equation (11) below to find the projection error. The process then minimizes the error in order to solve for the intrinsic parameters (5).

The world point $P(X,Y,Z)$ $X = (x, y, z)^T$ is projected to a point on the unit sphere $X_s = (x_s, y_s, z_s)^T$, and from the sphere to a point m on the image plane $m(u, v)$. K_c equation (5) is the intrinsic camera parameter matrix.

$$K_c = \begin{bmatrix} \gamma_x & s & u_0 \\ 0 & \gamma_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Where focal length is γ_x, γ_y, s is the image skew, and $p\ (u_0, v_0, 1)^T$ the principal point. The skew s is affected by both the focal length (γ) and the aspect ratio α. The imaging projection process is then captured by extending the single view point camera model $m=K_C[R\ t]X$ relating the space point matrix X, intrinsic parameters K_C, the extrinsic parameters R' and t to the image projection matrix $m(u', v')$ and are substituted in equation (6).

$$\beta m = K_c\left(\frac{R'X+t}{\|R'X+t\|}\right) + \xi e \tag{6}$$

where β is a scale factor, R' is a 3 x 3 rotation matrix, t is the vector of translation $t = (t_x, t_y, t_z)^T$ and ξ is the mirror shape parameter. The mirror is a paraboloid if $\xi = 1$, an ellipsoid or hyperboloid if $0< \xi<1$, and a plane if $\xi = 0$. R' is the mirror to camera rotation matrix, and t is the mirror to camera translation matrix as before.

The eccentricity e is a vector of the eccentricity in the x, and y direction and is a measure of how much the cross section deviates from being circular. The values and their interpretation are given below.

The eccentricity of a circle is zero.

- The eccentricity of an ellipse which is not a circle is greater than zero but less than 1.
- The eccentricity of a parabola is 1.
- The eccentricity of a hyperbola greater than 1.

The relationship between ξ and e is given in equation (7) and summarized in Table 11.

Both ***xi*** (ξ) and eccentricity (***e)*** were iterated during the calibration**.** ***xi*** (ξ) converged towards 1.0 and $\boldsymbol{e_x}$, $\boldsymbol{e_y}$ converged towards 1.0.

$$\xi = \frac{2 * e}{1 + e^2} \tag{7}$$

Table 11. Relationship Between Eccentricity e and Mirror Parameter ξ

	Ellipsoidal	Paraboloid	Hyperbolical	Planar
e	$0 < e < 1$	$e = 1$	$e > 1$	$\boldsymbol{e} \rightarrow \infty$
ξ	$0 < \xi < 1$	$\xi = 1$	$0 < \xi < 1$	$\xi = 0$

In our case the IR omnidirectional camera with parabolic mirror, the relationship between the world point ***(X, Y, Z)*** and the spherical point ***(r, θ, φ)*** are given by applying trigonometry as in Fig. 10 and as noted in equations (1) to (4), and resulting in equations (8), (9), and (10).

$$X = r \sin \varphi \cos \theta \tag{8}$$

$$Y = r \sin \varphi \sin \theta \tag{9}$$

$$Z = r \cos \varphi \tag{10}$$

The world point vector ***X*** (6) is replaced by its spherical coordinate equivalent elements from equations (8), (9), and (10) resulting in (11). The initial ***(r, θ, φ)*** points are substituted into (11) along with the estimated *Kc, R',* and *t* matrices and the projection point is calculated, the cost function *f(x)* is then minimized.

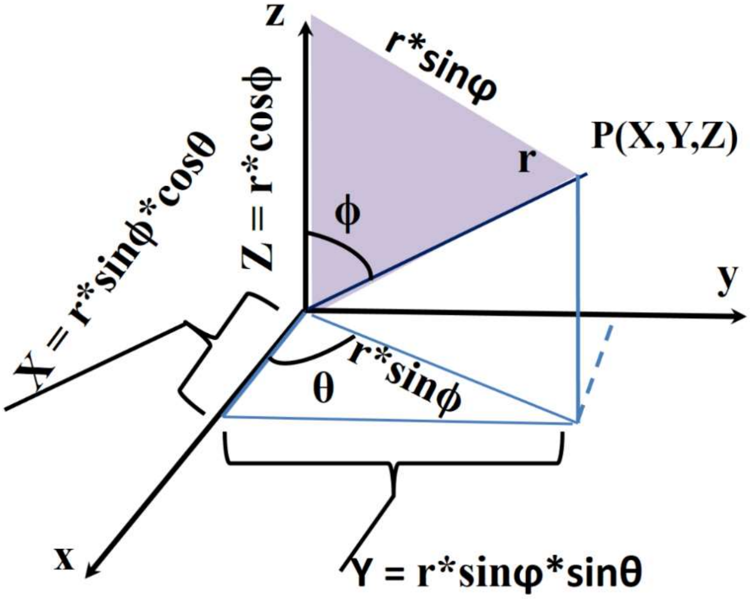


Figure 10. Spherical Coordinate Diagram:
Showing the relationship between Spherical Coordinates and Cartesian coordinates representing (8), (9), (10).

$$\beta m = K_c \left(\frac{R' \begin{bmatrix} r\sin\varphi\cos\theta \\ r\sin\varphi\sin\theta \\ r\cos\varphi \end{bmatrix} + t}{\left\| R' \begin{bmatrix} r\sin\varphi\sin\theta \\ r\sin\varphi\sin\theta \\ r\cos\varphi \end{bmatrix} + t \right\|} \right) \tag{11}$$

$f(x) = \frac{1}{2}\sum_1^n [P(x_i) - e_i]^2$, where ***n*** is the number calibration boards, ***P(x_i)*** is the projected calibration board center, and ***e_i*** is the extracted calibration board center. The cost function is the Euclidian distance between the projected calibration board center point and the extracted value in the image. The n calibration board re-projection points are then used to triangulate the camera center. Equation (11) represents the calibration equation directly in the ***(r, θ, φ)*** framework, and is used to find the re-projected calibration points, and the fitness function *f(x)* is then minimized using Levenberg-Marquardt approach. It is the relationship between the intrinsic, extrinsic parameters and the camera geometry.

STEP 3: MEASURE THE GRID CORNER (4 OUTSIDE) INTERSECTION POINTS.

Our method of selecting the four outside corners as a starting point and applying our algorithm to find the position of the calibration board center worked significantly, on the order of 40X, better than the baseline methods. The baseline methods saw errors on the order of 81 pixels compared to ours at 2 pixels. The baseline methods struggled with the low resolution IR images which either introduced larger error, or failed completely to converge. The calculated coordinates of the center of the calibration board was then used to directly calculate the spherical coordinates and the projected radius to the center of the camera was found by minimizing the re-projection error.

The four corners of each calibration grid in the data set images, Fig. 11, are clicked on by the user with a corner assist algorithm, and the pixel coordinates of the corners are extracted and the central point of the grid (green square at the center of the distorted corners box) is approximated by geometry of the four image points.

We first attempted the MATLAB automatic corner selection [32] used in [31], but this method broke down on our lower quality IR images necessitating our modified approach. The internal extrapolation failed and the internal points being out of position resulted in the error solution going out of bounds in some of the images. This necessitated us to estimate the center from the geometry of the four outside corners which worked much better. The image coordinates of the entire calibration board center points are then used to find the θ, φ, and r value of the calibration board centers which are then substituted into (11) and the difference between the projected points and the observed image points are minimized. θ, and φ are the angles to the center of the capture board grid observed from image coordinates, and r is the distance from the center of the camera to the center of the capture board in pixel coordinates obtained from equations (1-4). The value of r is found by $r = sqrt(x_p^2 + y_p^2)$, and x_p and y_p are the pixel coordinates of the grid center in the image.

The θ, φ and r values are substituted into (11) from step 2.

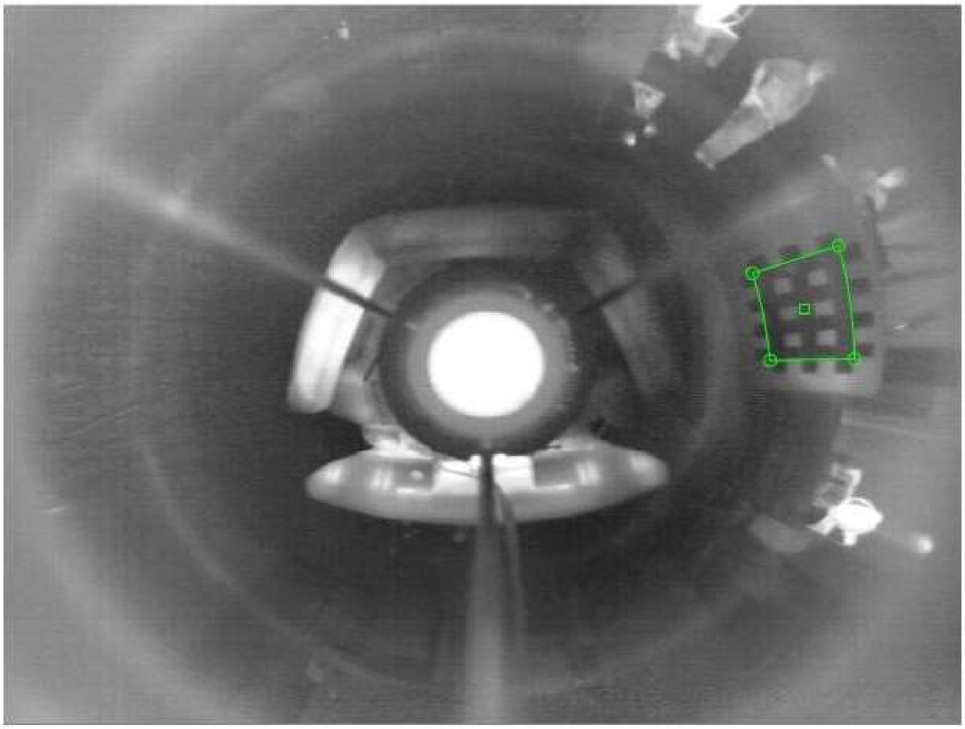


Figure 11. Calibration Board Capture
Calibration Board center approximation using the four outside corners (circles) the center shown by the green square is found.

STEP 4: SOLVE FOR THE CAMERA INTRINSIC PARAMETERS (KC).

The $\boldsymbol{K_c}$ matrix (5) is then solved for and the results displayed. The $\boldsymbol{(r_c, \theta_c)}$ is re-projected on the image and the error is minimized to find the solution to (11). This DSC direct approach has the best calibration results when compared to the Baseline1 (distortion) and 2 (single viewpoint spherical) methods, but with smaller error. Baseline 3 (generic) had much higher error and went out of bounds on the dense data set.

STEP 5: SOLVE FOR THE CAMERA EXTRINSIC PARAMETERS.

The calibration process then finds the $\boldsymbol{R'}$ and $\boldsymbol{t}$ matrices such that a least square error function is minimized to best match the mapping, and the process iterated until convergence or threshold is met.

Table 12. Five Process Steps for DSC

Steps	Step Details
Step 1	Establish a thermal calibration grid 2D Planer Pattern with thermally reflective grid of known geometry, which consisted of five rows of seven squares or a total of 48 corner points. (needed for Baseline methods) The grid was constructed on a white board with a cross pattern of silver reflective tape. The calibration boards were then used at different angles and distances from the camera. (DSC only used the outside corners)
Step 2	Direct Spherical Calibration (DSC) Establish the DSC Camera Model in terms of ***(r, θ, φ)*** $\beta m = K_c \left\{ \frac{R'\begin{bmatrix} rsin\varphi cos\theta \\ rsin\varphi sin\theta \\ rcos\varphi \end{bmatrix}+t}{R'\left\lVert\begin{matrix} rsin\varphi cos\theta \\ rsin\varphi sin\theta \\ rcos\varphi \end{matrix}\right\rVert+t} \right\} + \xi \mathrm{e}$
Step 3	Measure the grid corner (4 outside) intersection points at different unknown positions which are related to the sensor coordinate system by a rotation matrix ***R' = [R'*** R'_x, R'_y, R'_z***]*** and translation matrix ***t = [*** t_x, t_y, t_z ***],*** the extrinsic parameters
Step 4	Solve for the camera intrinsic parameters Kc $K_c = \begin{bmatrix} \gamma_x & s & u_0 \\ 0 & \gamma_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$ ***Where s is the skew and is a function of α*** the camera aspect ratio and the focal length, ***γ*** is the focal length, and u_0, v_0 is the principal point.
Step 5	Solve for the camera extrinsic parameters $R' = (R'_x, R'_y, R'_z)^T$, R' is the camera rotation about the ***x, y, z*** axis. $t = (t_x, t_y, t_z)^T$, ***t*** is the translation of the camera along the three ***x, y, z*** axis.
Step 6	Iterate steps 2 - 5 until convergence since the intrinsic and extrinsic parameters influence each other.

STEP 6: ITERATE STEPS FOUR AND FIVE UNTIL CONVERGENCE.

Since the intrinsic and extrinsic parameters are related we iterate steps four and five until the minimized re-projection errors converge. The six steps are summarized in Table 12. The results of our DSC method and the three Baseline methods will be presented in Section 2.4.2.

3.1.3. Optimal number selection of image capture of calibration boards

We will use Pareto optimization as an analysis tool to find the optimum calibration approach and number of calibration board images. This section determines the effect of the number of calibration board images on the calibration results. The methodology used to find the optimum number of calibration capture boards is to find the optimal calibration condition when minimizing the error during the determination of the intrinsic camera matrix K_c (5), and also minimizing re-projection error. To accomplish this optimization, we will use a MATLAB Pareto optimization to find the min. function that best satisfies error, ratio of calibrated to actual and time minimization.

Fig. 11 describes the Pareto optimization concept which is for the minimization of multiple objective factors. In an optimization problem it is typically the evaluation of decision variables by minimizing a cost function.

However, in most real life optimization problems the cost function is multi-dimensional trying to satisfy multiple objectives. This is our case, the example shown in Fig. 8, the two cost objectives are re-projection error ε_d and time t. The Pareto Frontier curve shown represents the set of solutions that satisfy the minimization of both constraints according to the fitness function f. In the Fig. 12 example $f(\varepsilon_{d1}, t_2)$ as well as $f(\varepsilon_{d2}, t_1)$ are both optimum solutions, and lie on the pareto optimized frontier. We will use a three objective fitness function to simplify the Pareto optimum frontier, where the three objectives are the ratio of calibrated to actual (R_{CA}) of distance and angle distribution in the polar coordinate, re-projection error (ϵ_d), as obtained from the calibration process, and process time (t). While time (t) is not important from an operational viewpoint since the calibration is done off-line, it was chosen as a comparison parameter between the three methods.

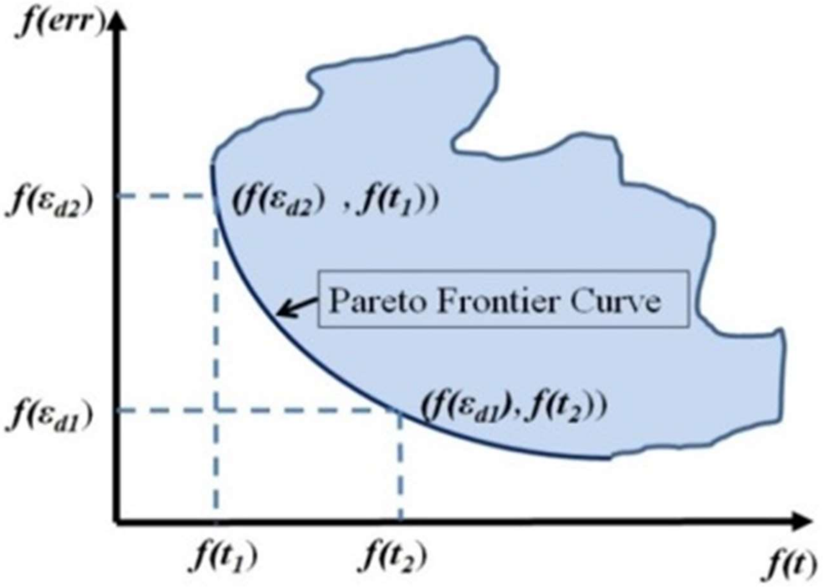


Figure 12. Pareto Optimization Diagram:
Showing an example Multi-objective optimization space and the Pareto frontier curve representing the loci of points satisfying both objectives (ε_{di}, t) optimal number selection of image capture of calibration boards.

The Pareto optimization would find all points ***$(R_{CA}, \epsilon d, t)$*** in the objective space that satisfy our scalar fitness function, equation (12).

$$Arg_{n_{min}} = f(R_{CA}, \varepsilon_d, t) = \sqrt{(|R_{CA} - 1|^2 + \varepsilon_d{}^2 + t^2)} \tag{12}$$

We used the MATLAB Pareto Optimization Tool Box to find the global optimum for the four methods and data sets. Ratio of Calibrated to Actual data is given as ***R_{CA}***. An example of the concept is shown in Fig. 13 with perfect desired ***R_{CA}*** given as the blue circle with ***$R_{CA} = 1.0$***. The red curve represents the actual ***R_{CA}*** and varies around the perfect ratio of ***$R_{CA} = 1.0$***. The ***R_{CA}*** is given by equation (13),

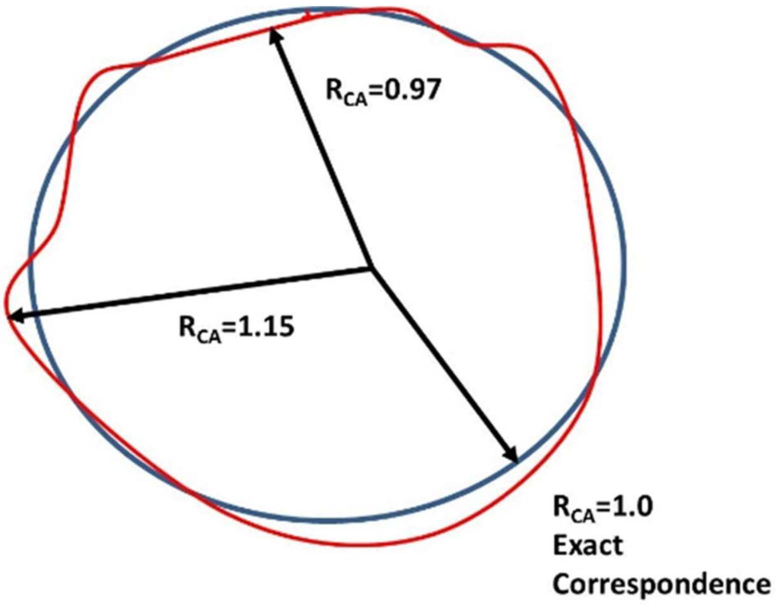


Figure 13. Ratio Calibrated to Actual:
Ratio of calibrated to measured data as a shape factor, where RCA =1.0 is perfect correspondence and is represented by the blue circle. The red curve is the actual variation RCA about the blue circle.

$$R_{CA} = \frac{(R_R + R_A)}{2} \tag{13}$$

Equation (13) is used as a common shape factor. Where $\boldsymbol{R_R}$ is the ratio of calibrated to measured radius and $\boldsymbol{R_A}$ is the ratio of calibrated to measured angle.

The evaluation criteria for $\boldsymbol{R_{CA}}$ will be the absolute value of the difference between the ratio of calibrated to measured ***(R_{CA})*** and the perfect correspondence of 1.0 as $|(\boldsymbol{R_{CA}})\boldsymbol{-1}|$. The error parameter is the mean of the re-projection error as a function of the diameter of the plotted re-projection circle. $\boldsymbol{\varepsilon_d}$ is given by equation (14) as a re-projection error.

$$\varepsilon_d = mean\left(\left|sqrt\left(d_{actual} - d_{re-projecte}\ \right)^2\right|\right) \tag{14}$$

The last parameter time ***(t)*** is the period during the calculation of the calibration modules proposed in Section IV.B and is given by ***t = (time at finish – time at start)***. The relationship between the Ratio of Calibrated to Actual ***(R_{CA})***, error ***(ε_d)***, and time ***(t)*** will be explored in the experimental results section.

3.1.4. Comparison to other methods - Baselines 1,2,3

In this chapter our method plus three existing methods for identifying checkerboards on calibration images were analyzed. The four methods are compared in Table 13.

Table 13. Methodology Comparison

Criteria	Baseline 1 distortion [7-8]	Baseline 2 single viewpoint spherical[9]	Baseline 3 generic [36-37]	DSC
Model	Distortion	Spherical	Distortion	Spherical
Points Re-projected	25 x 48	25 x48	25 x 20	25 x 1 (centroid)
Mirror	Spherical	Spherical	Spherical	Spherical
Approach	Poly-nominal Approx.	Single Viewpoint	Generic	Direct Spherical

From the three, two Baseline methods were chosen as the best candidates to modify for the O-D IR camera calibration, and the best methods of both were combined with our methodology for an improved method for calibrating IR omnidirectional cameras. An exhaustive comparison of other methods was not the purpose of this work. The methods chosen then served as the starting point for the adapted and improved DSC method described in Section 2.4. Due to the quality of thermal images the edge boundaries were not as crisp as visual images. As a result, the automatic corner finders had marginal performance. Often all three comparison methods would provide poor results when the corners were poorly selected even in the manual mode with corner assist. This required extensive manual tuning in the Baseline 1 (distortion) and 3 (generic) methods. The Baseline 2 (single viewpoint spherical) method used the four outside corners as a starting point but still calculated the internal grid points which took more time and was prone to error with the IR images if the points weren’t well chosen. In the dense data set case all three baseline methods had trouble with the partially obscured calibration boards due to the camera structure. This caused an extensive amount of rework making these methods not practical for the lower quality IR images

The DSC direct spherical method used the four outside corners of the calibration grid to find the calibration board center point and this pixel spherical coordinate was then used to directly calibrate using the one spherical point for each calibration board. The geometric center of the whole grid is then calculated from these four geometric points and the spherical coordinates of the center of the grid is used in the calibration. This avoids the errors introduced by finding a best fit of the line matching of the individual grid square elements (which does not work well in the IR case) to establish the calibration geometry. Our results show that this improves the calibration accuracy, and requires less data entry time. The method also has the best computational performance.

3.2. Technical Approach for Index and Region Segmentation for Sensor Fusion of Omnidirectional Far-infrared Camera and Visual Camera for Vegetation Detection

The fusion of the O-D Color visual and O-D IR data was used to enable the robotic perception system to adapt to different lighting and environmental conditions. In this chapter we focus on the fusion of O-D far-IR and visual stream with the System approach shown in Fig 14.

We compare our results to [89-91] in section 3.4 Experimental Results. [89] presents a Compressive Data Fusion approach for multi-sensor fusion. [90] studied land cover changes using Normalized Difference Vegetation Index from 2005-2015 to assess impact of changes in

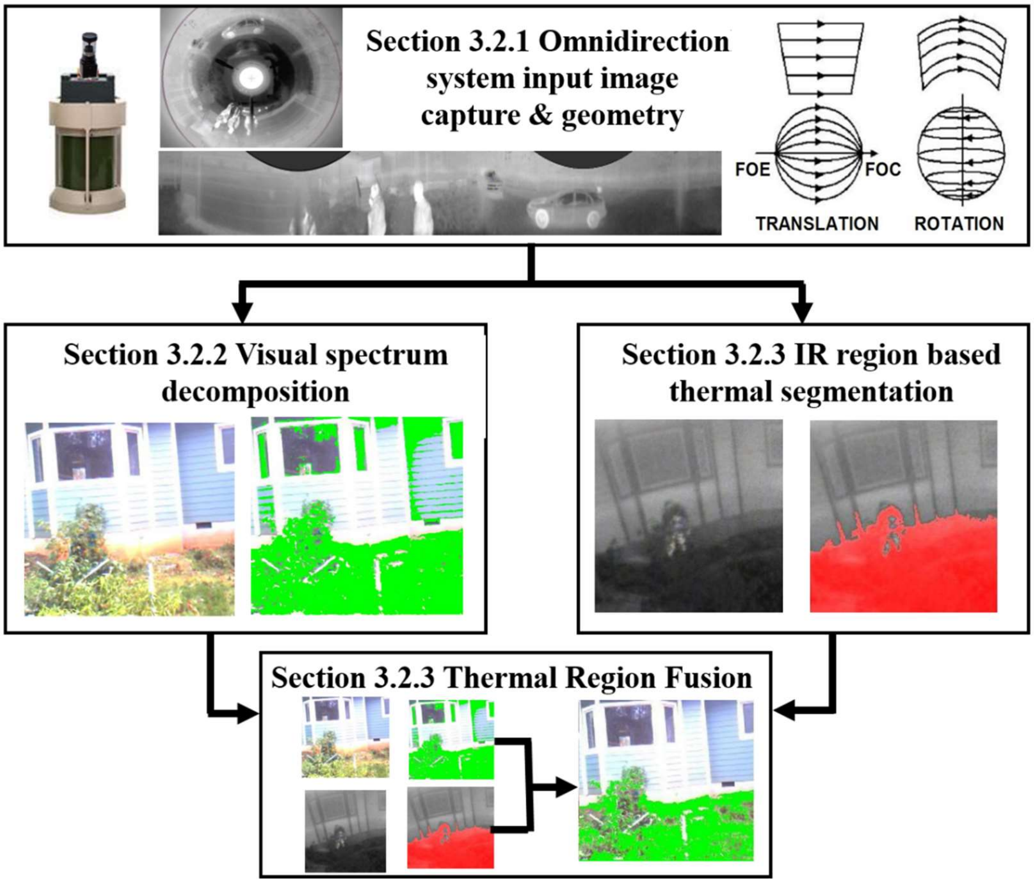

Section 3.2.1 Omnidirection system input image capture & geometry
FOE
FOC
TRANSLATION
ROTATION
Section 3.2.2 Visual spectrum decomposition
Section 3.2.3 IR region based thermal segmentation
Section 3.2.3 Thermal Region Fusion

Figure 14. Baseline Fusion Block Diagram

Technical Approach consist of the following subsections: 3.3.1 Omni- Direction Camera Setting, 3,3,2 Visual and IR Index Based Vegetation Detection, 3.3.3 IR Stream Segmentation using Region based Thermal Analysis, and Thresholded Region Fusion (TRF).

watershed on ground cover. Results included shrub 0.67-0.737, secondary forest 0.737-0.804, and primary forest 0.804-0.876. [91] used a UAV to gather Lidar and Hyperspectral data. The Lidar point cloud determined canopy structure and the hyperspectral identified species specific spectral signatures. The data was fused to achieve overall 0.76 accuracy. [92-94] show results for false positives (FP) ranging from 32.5% -61.29% with FP peaks as high as 77.4%.

Our proposed system architecture applies texture and frequency analysis (segmentation and signature) to images from low cost, O-D visual and IR sensors to provide object detection, classification, and tracking. The final approach is the fusion of visible and IR perception systems using O-D sensors. Each block in the Fig. 14 diagram represents the subsections as follows:

- Section 3.2.1 discusses the O-D Camera System and the O-D Coordinate System setting.
- Section 3.2.2 develops the computational foundation for a new approach to multi-spectral sensor fusion that identifies semantically significant object classes based on spectral and thermal signatures using visual and IR spectrum decomposition, finally.
- Section 3.2.3 discusses the method for IR region segmentation using region based thermal threshold analysis, and discusses the TRF method.

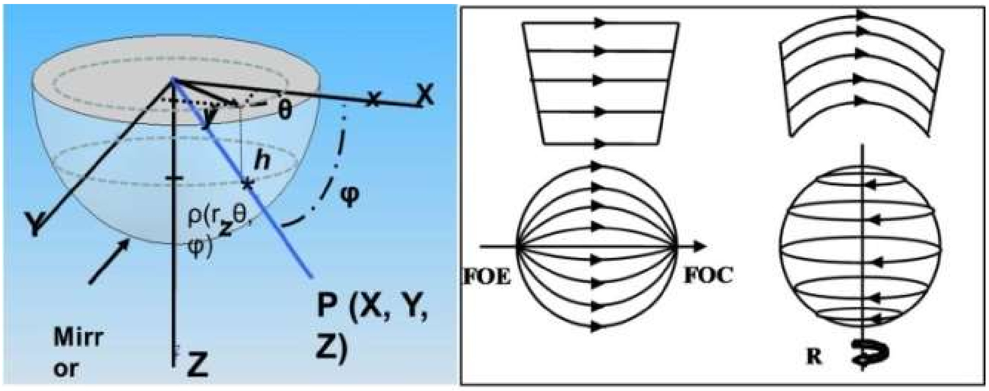
3.2.1. Omni-direction Camera Setting

Using spherical reflecting surfaces, a projected 360-degree field of view is created and transformed into a single 360 linear plane. With a single Omni-vision approach, reconstructions of scenes create a reduced set of unknowns in the equations as compared to the many camera approach. With improved Omni systems, the corrections required are reduced to create 360 fusions, the computation requirements are reduced and sources of error are improved for electro-optical cameras. Additionally, multi-spectral sources can be fused with reduced error and the calculations errors for sensor fusion does not propagate forward in the perception system world model.

The transformational relationship between rectangular coordinates and spherical coordinates was shown in Section 2.3.1 Fig. 2 and described in the equations (1-4).

In O-D optical flow the difference between flow fields in rectangular space and spherical space can be used to extract semantics from the O-D image. Translation in rectangular space maps to arcs in the spherical space in the direction of flow and is why O-D vision is useful for determining ego-motion. Note that translation has focus of expansion (FOE) and focus of contraction (FOC) points in the image where rotation does not. Translation flow is from the Focus of Expansion (FOE) to the Focus of Contraction (FOC). This motion can be calculated. The rotation motion is preserved as circles on the surface of the mirror in the spherical coordinates. Vertical lines in the rectangular space map to radial lines in the spherical image space.

In Fig. 15, x and y are the image plane coordinates of the pixel, P is the point in space being observed by the O-D camera, and the parameter z traces the surface of the mirror. In the spherical coordinate system, θ and φ define the direction of the ray and r is the distance to the origin of the mirror. The orientation θ ranges from 0° to 360° around the edge of the mirror, and the pitch φ, ranges from 0° when pointing straight down, to 90° when pointing at the horizon. The calibration parameter h and the calibration process were presented in [1] where the camera was calibrated.



a) Geometry Relationship b) Translation c) Rotation

Figure 15. Geometry Relationship
a) geometry of the O-D camera, b) Translation, c) Rotation.

3.2.2. Visual and IR Index Based Vegetation Detection

The classical vegetation detection looks at red color bands in the visual spectrum and compares this to near-IR spectrum. The NDVI feature works well in chlorophyll rich vegetation, but doesn't do as well in dry vegetation or desert scenes. It also was a problem when operating in new areas that were not trained. Our approach hopes to use the thermal threshold for vegetation fused with the color vision modified NDVI (MNDVI) to improve performance in dry vegetation and false detects.

Our approach modifies the NDVI to look at the red band and far-IR band difference and fuse this with the thermal characteristics of the vegetation. Vegetation is detected based on the ratio of red reflectance and far-IR reflectance. This technique allows us to take advantage of the physical properties of vegetation and how they reflect and absorb light. This is influenced both by the absorption of the chlorophyll and the water content in the vegetation. These techniques apply the band ratio of the far-IR bands and red band in the visual range. The water content of vegetation absorbs light above 1400 nm, and the chlorophyll absorbs the red and blue bands leaving the green band. The result is that vegetation shows a high reflectance in the green and IR bands. Several vegetation indices have been developed; these are:

The Normalized Difference Vegetation Index (NDVI) is given as (15), which is the ratio of the difference and sum of the NIR reflectance and the RED reflectance.

$$NDVI = (I_{NIR} - I_{RED}) / (I_{NIR} + I_{RED}) \tag{15}$$

Several other indices are also used such as the Difference Vegetation Index (DVI) as $DVI = I_{NIR} - I_{RED}$, which is the difference between NIR reflectance and the RED reflectance. The Perpendicular Vegetation Index (PVI) is given as $PVI = \sin(\alpha)\, I_{NIR} - \cos(\alpha) I_{RED}$. Where PVI is the difference between the NIR reflectance times $\sin(\alpha)$ and RED reflectance times $\cos(\alpha)$, and represents the perpendicular distance from the NIR reflectance point to the soil line. The angle alpha (α) is the angle between the soil line and the near infrared reflectance axis.

We use the Normalized Difference Vegetation Index (NDVI), where $NDVI = (I_{NIR} - I_{RED}) / (I_{NIR} + I_{RED})$ modified for far-IR (16).

$$MNDVI = (I_{IR} - I_{RED}) / (I_{IR} + I_{RED}) \tag{16}$$

The MNDVI was fused with the region based IR signature explained in section 3.3.3. to enhance the semantic segmentation of the IR/Vision stream. We are using potentially low cost IR and visual sensors to achieve multiband spectral signatures. Fig. 16 shows the location of a point of interest in the O-D IR sensor, unwrapped image and extracted IR and visual images.

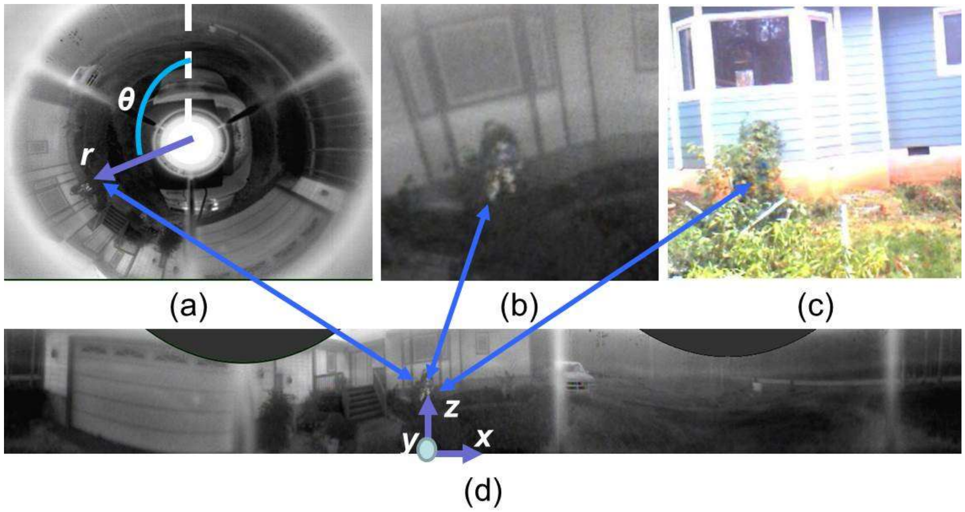
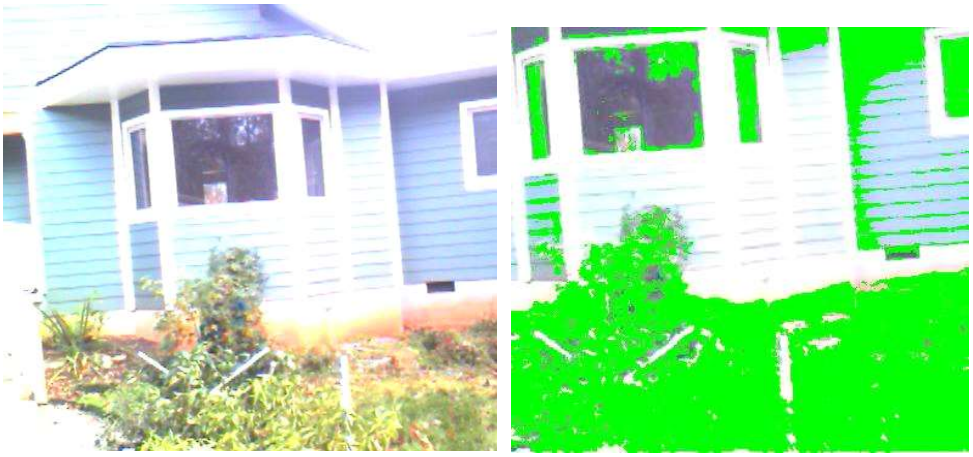


Figure 16. Spherical-Cartesian Coordinates
The O-D geometry, (a) shows 360-degree O-D image with the r, θ geometry overlaid. (b) The cropped IR region of interest, (c) the visual cropped region of interest (d) shows the unwrapped rectangular image, and the X, Y, Z geometry axis.

We looked at multispectral bands and region based IR spectral analysis to better characterize the spectral signature of different materials. This work looks at the effectiveness of the MNDVI and Region Based IR method as compared to the classical NDVI approach to more fully characterize the signatures of vegetation and other materials. The indices value is then used as a threshold to detect vegetation. There are issues with false detects in the areas where the bands overlap when using just the MNDVI. We used the MNDVI approach on our data in Fig. 17 below to show the effect of this approach and the number of false detects.



(a) (b)

Figure 17. MNDVI Approach

MNDVI Approach: (a) Original image, (b) Processed image using MNDVI vegetation index approach. This result has a relatively high number of false detects.

We as humans are accustomed to using color to distinguish materials. The color and reflectivity are important indicators of the material composition of an object.

The spectral signature of the material or material reflectance spectrum *ρ(λ)* is related to the reflected light or spectral radiance *Ls(λ)* and the impending scene radiance *Li(λ)* by the following (17):

(17)

This can be rearranged into the reflectance spectrum *ρ (λ)* in terms of the ratio of spectral radiance *Ls (λ)* to the scene radiance *Li (λ)* as given in (18)

(18)

$$\rho(\lambda) = \frac{L_s(\lambda)}{L_i(\lambda)}$$

$$L_s(\lambda) = \rho(\lambda) * L_i(\lambda)$$

This wavelength ***λ*** is filtered into ***k*** spectral bands. Fig. 18 shows a representative Cumulative Density Function (CDF) for the three primary color bands (red, blue, green) in the visible spectrum. The 360 color camera is the humingbird360 with a Point Gray Flea3 with 12 bits and 8.8-megapixel camera. The CDF is defined as the accumulation of pixels in the histogram of each color band.

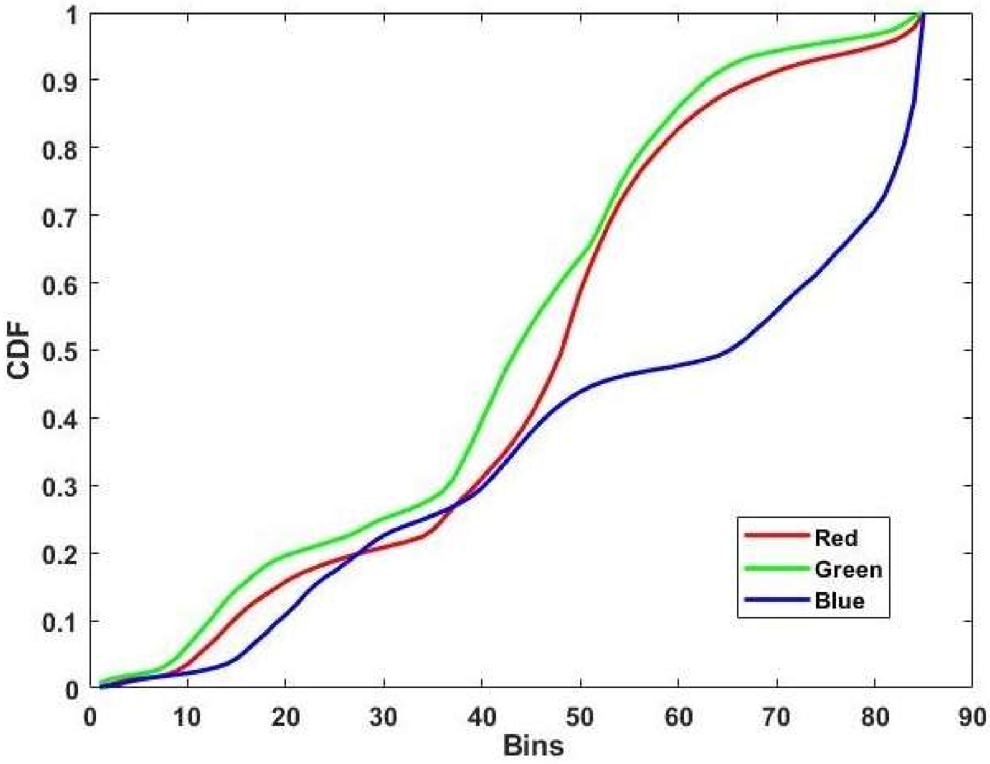


Figure 18. Cumulative Density Function

Representative Cumulative Density Function (CDF) for the three color bands in the image is a summation of the probabilities of each intensity at the different wavelengths

The approach we are developing is to fuse the red band visible and IR region spectrum and utilize these spectral signatures to extract scene semantics from the O-D IR and visual images. We demonstrate that each material has a characteristic visual and thermal spectral signature from which to classify the material. We then compare the MNDVI red band approach to the combined MNDVI red band and IR region based thermal signature approach, and finally compare the fused results to the baseline MNDVI indexed based approach.

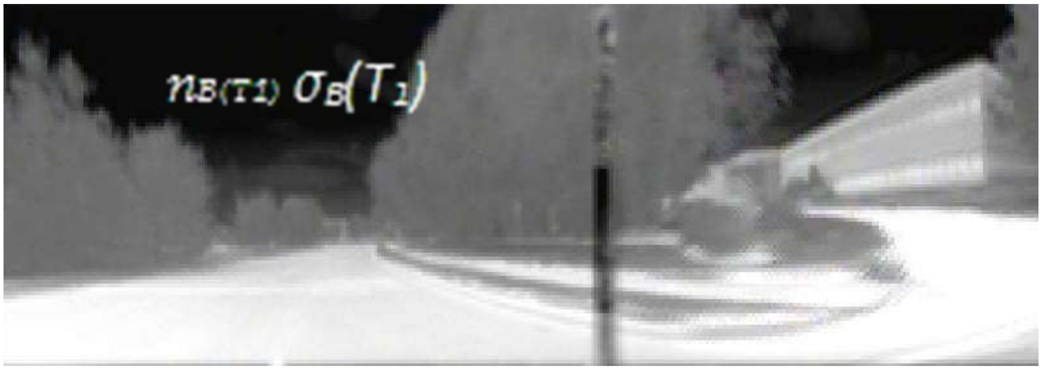
3.2.3. IR Stream Segmentation using Region based Thermal Analysis

To address the issue of false detects in the index based approach, semantic extraction based on thermal regions will be investigated to accentuate vegetation detection and reduce the number of false positives.

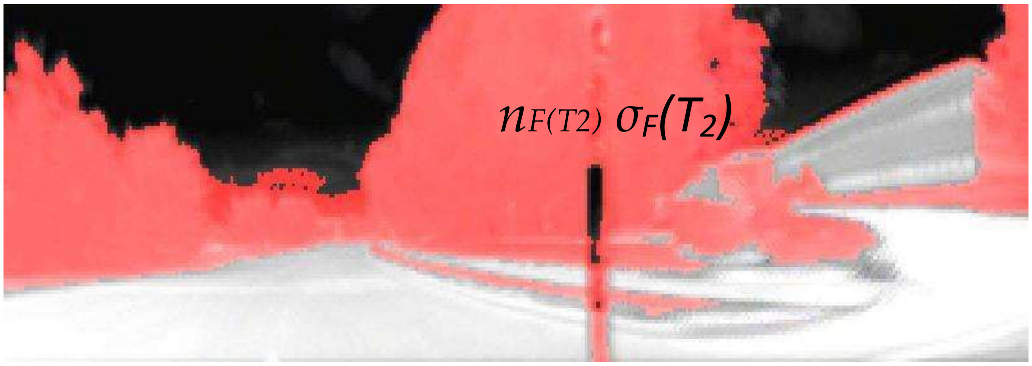
Fig. 19 is the IR region segmentation using region based thermal segmentation analysis 19) a is the original image, 19) b shows that all of the vegetation is at the same temperature, and 19) c shows that the hard concrete road and sidewalks are at a different temperature. The sky is dark and has not absorbed any temperature.

Fig. 19 shows the region based segmentation of the IR image. The areas of similar gray level (same temperature) will be segmented and matched to the spectral signature for that region and then classified based on the spectral content and thermal characteristics. For instance, the sky is dark with no thermal content, the vegetation is dark gray due to its water content causing it to pick up less heat than the sidewalks or pavement, and the pavement and sidewalks are white due to picking up heat during the day. These thermal based material regions can then be correlated with the material spectrum to aid in identifying the material.

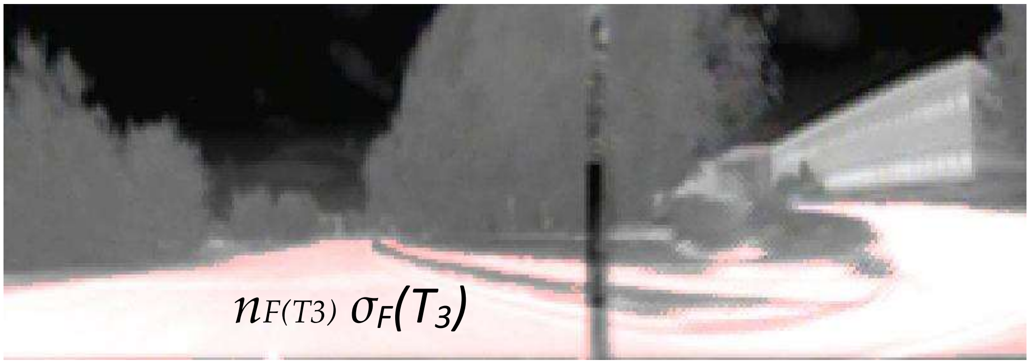
Fig. 19) a is the original unwrapped thermal image, 19) b shows the vegetation region segmented and has the formula for the region over laid. The sky is the background and has the background CDF times the background variance squared at threshold T_1, and the vegetation has the Foreground CDF and foreground variance at threshold T_2. Fig. 19) c shows the pavement and sidewalk regions segmented at Foreground CDF and Variance at threshold T_3. After segmenting the various material regions and extracting the spectral signature for the regions, the two results will be fused by the union of the two sets.



a) Original IR Image and Region 1 background = sky



b) Region 2 Vegetation Region all at same temperature



c) Region 3 Sidewalk and road at different temperature

Figure 19. IR Region Segmentation
IR Segmentation using region based Spectral Analysis (a) is the original image, (b) shows that all of the vegetation is at the same temperature, and (c) shows that the hard concrete road and sidewalks are at a different temperature. The sky is dark and has not absorbed any temperature.

The thermal image has similar materials clustered in the same region of the IR thermal space. At different ambient temperatures these materials will shift in color but will remain clustered. The thermal signature of certain terrain types and vegetation will have different emissivity, but the behavior is characteristic of the material. By extracting this region based signature from the thermal behavior and fusing it with color multispectral signature the different types of vegetation and terrain can be classified. During the day vegetation is cooler than the soil and is darker in the IR image. At night vegetation and soil reverse and the vegetation is lighter. Personnel and vehicles that are emitting heat can easily be picked out of the thermal image. They are also visible in the IR image when the visual is obscured by dust or fog. These attributes are an advantage of using the thermal information

Using the O-D IR image we performed region based spectral analysis to group like materials together, and extract region based semantics from the IR signature. This is used with the multi-spectral signature and thermal map thresholds to identify the semantics of the scene.

Representing the gray scale of the IR image as histograms, we partitioned the image (segmentation) into homogeneous regions representing areas of similar temperature. During the day the ground and other solid objects absorb heat faster than vegetation so the vegetation areas will be lighter than the ground. At night this reverses and the foliage will look darker. Considering this grayscale image, we let the number of times gray level ***i*** occur as n_i. The probability ***P*** that a pixel at ***(r, θ, φ)*** will be at level ***i*** is given as (19).

$$P_{r,\theta,\varphi}(i) = \frac{n_i}{n}, 0 \le i < l \tag{19}$$

where ***l*** is the total number of gray levels in the image, and n is the total number of pixels. So the probability of intensity ***i*** at a given ***(r, θ, φ)*** location is the ratio of the number of pixels at intensity ***i*** or n_i to the total number of pixels' ***n*** $\Pr_{\theta,\varphi}(i)$ is the image histogram for pixel ***i***, the value is normalized to ***[0,1].*** We define the Cumulative Density Function (CDF) as (20)

- Conditional Local Point Statistic

- 2D-3D Feature Fusion
- Laser Remission
- NDVI-Laser Fusion
- SVM Multi-Cam Fusion
- Multi-Cam NDVI Fusion

$$CDF_{r,\theta,\varphi}(i) = \sum_{J=0}^{I} P_{r,\theta,\varphi}(j) \tag{20}$$

The CDF is the summation of the probabilities for the intensity at each pixel. Using (21), for optimal thresholding, to segment regions we achieve a mapping of regions of like thermal intensity. Different materials absorb heat differently and thus we can use this in conjunction with the Spectral signature to determine the regions of similar material.

$$\sigma_{within}(T) = n_{B(T)}\sigma_B{}^2(T) + n_F(T)\sigma_F{}^2(T) \tag{21}$$

where ***η_B*** is the background histogram CDF,

$$n_B(T) = \sum_{J=0}^{T-1} P_{r,\theta,\varphi}(j)$$

where ***η_F*** is the foreground histogram CDF.

$$n_F(T) = \sum_{J=T}^{N-1} P_{r,\theta,\varphi}(j)$$

σ_B is the variance of the pixels in the background ***(<T)***

$$\sigma_B(T) = \sqrt{(h_P - h_{Avg})^2}$$

σ_F is the variance of the pixels in the foreground ***(>T)***

$$\sigma_F(T) = \sqrt{(h_P - h_{Avg})^2}$$

where h_p is the histogram value at the pixel of interest and h_{Avg} is the average value of all the pixels above the threshold for foreground or below the threshold for background. The optimum threshold for Fig. 18, 19, and 20 is determined by a combination of the triangle threshold method and the optimization for T of equation (21) plus the statistical extent of the thermal IR region.

3.2.4. Genetic Algorithm

A genetic algorithm was applied to find the optimal threshold and the optimal clustering of thermal regions. The genetic algorithm is an optimization process that is based on natural selection, and selects the fittest individuals for reproduction of the next generation. It is an adaptive heuristic search algorithm that selects the best parents from which to generate the children based on a fitness cost function. The algorithm modifies the next generation based on the genes of the parents that are randomly mutated. Each generation the process selects the best parents based on a clustering cost function with an upper and lower threshold. The thresholds are optimized to cluster the regions. The genetic algorithm ran for 50 generations and generated 10,400 mutations.

Equation (21) is used as input to the fitness function for a genetic algorithm which optimizes the regions of interest in the given image. For a starting T the standard deviation and variance of the foreground and background histogram is calculated. The CDF above and below that threshold is also calculated. These are then combined in equation (21) to find the standard deviation of the region of interest. This is tested against the thermal band for vegetation and iterated using the genetic algorithm until the optimum thresholds for the vegetation region are found.

3.2.5. Thermal Region Fusion

The overall process is shown in Table 18. The two algorithms will be applied and the results fused using the TRF approach:

- MNDVI index based modified for far-IR vegetation detection.

- Region based thresholded thermal segmentation augmented with a genetic algorithm to optimize the selection of the region of interest.

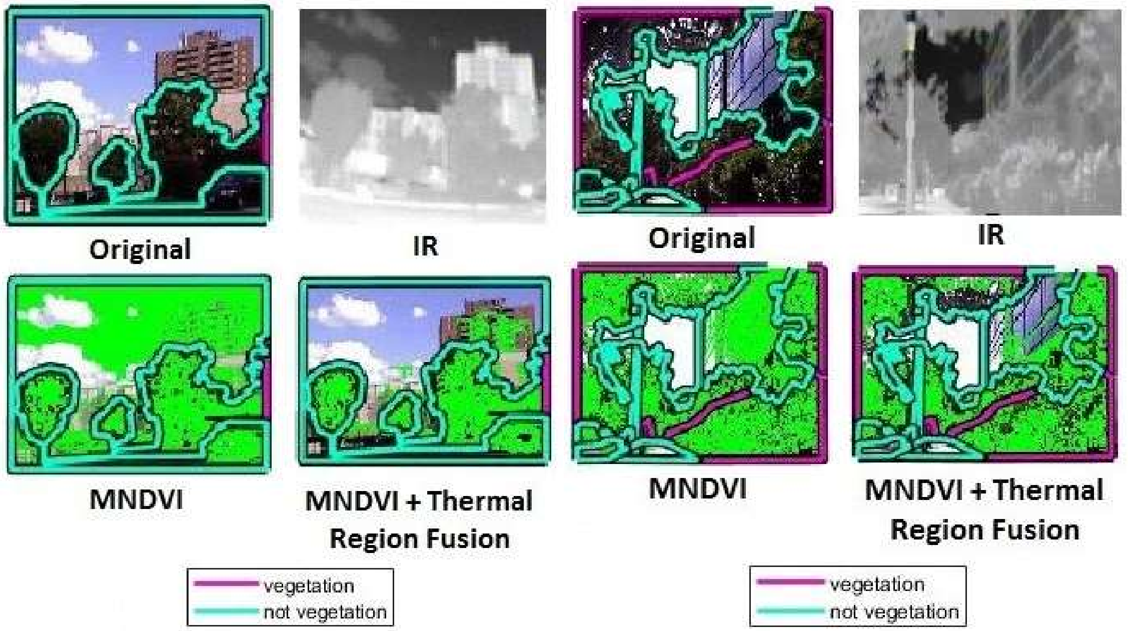
The experimental results are presented below in Chapter 4 Section 4.3.

Table 14. Proposed TRF Algorithm: Sensor Fusion of MNDVI Vegetation Index and Region Based IR

Steps	Step Details
Step 1	Apply the MNDVI vegetation index to extract regions of vegetation $MNDVI = (I_{FIR} - I_{RED}) / (I_{FIR} + I_{RED})$
Step 2	Apply thresholding to the IR image to identify regions of similar temperature $\sigma_{within}(T) = n_{B(T)}\sigma_B{}^2(T) + n_F(T)\sigma_F{}^2(T)$
Step 3	Apply region growing using a genetic algorithm to find all of the areas with the same temperature within the optimum threshold band.
Step 4	Fuse the MNDVI vegetation highlight with the IR region based thermally detected vegetation area to minimize false detects in the MNDVI signature
Step 5	Compare the vegetation index based results, with the combined MNVDI and Thermal Region Threshold Segmentation Fusion results.

Fig. 20 shows the fusion of the region in 20) b representing the vegetation thermal signature with the MNDVI result in 20) c. It still has some false detects but they are significantly reduced. Fig. 20 second row is the fusion of the MNDVI result with the thermal region representing the non-vegetation region. In this second application of the algorithm, this time the non-vegetation thermal region 20) f is fused with the MNDVI result 20) g.

Fig. 21 is a comparison for two original images a) and b), and for each image the following are shown: the original image with ground truth labeling, the IR image, the MNDVI image, and the Thermal Region Fusion (TRF). The images show the false detects in the MNDVI image and the reduction in those false detects in the TRF image. We used the MIT LabelMe website [47] to label the vegetation and non-vegetation regions shown in Fig. 21.



a) Original image 1, O-D IR excerpt, MNDVI and TRF results.

b) Original image 2, O-D IR excerpt, MNDVI and TRF results.

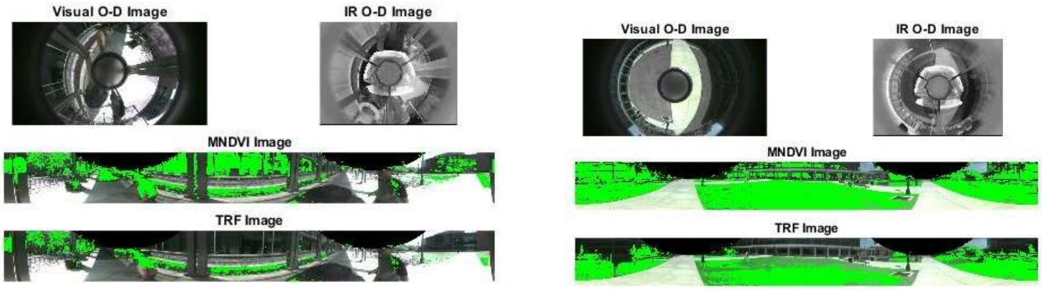
Figure 20. MNDVI and TRF Comparison O-D and Kinect

Two original images a) and b) and for each image the following are shown: the original image with ground truth labeling, the IR image, the MNDVI solution, and the Thermal Region Fusion. The vegetation detection and false positive reduction for the IR and Kinect visual images are shown.

Utilizing (11) along with the triangle threshold method to find the threshold for the vegetation region, (11) is minimized to find the extent of the region. We also applied a Genetic Algorithm to find the region of interest. We first apply the modified vegetation index using far-IR instead of near-IR. We next apply thresholding and region growing techniques to identify the regions of vegetation and non-vegetation. The results of the MNDVI and thermal region extraction are then fused to minimize false detects and the results are presented.

The contribution of this work is a computational foundation for a new approach to multi-spectral sensor fusion that identifies semantically significant object classes using sensor fusion of the O-D vision (Electro-Optical) and O-D IR sensors, applying the two streams into a thresholded segmentation and fusion genetic algorithm for the fusion of the IR and visual Spectral content to extract the regions of vegetation and other materials. Our algorithm applies both a modified segmentation Feature Extractors into the input fusion / classification. We will evaluate the TRF architecture against the baseline index based vegetation detection.

Fig. 22 shows two different images of different scenes with the original O-D visual and O-D IR image shown for each case. The unwrapped O-D images are then shown with the MNDVI and TRF detected vegetation regions overlaid. In each case the MNDVI image shows significant false detects. In Fig. 22) a the TRF method shows good vegetation detection and also has good false positive rejection. However, in Fig. 22) b while the false positive rejection is still good, the vegetation detection is less accurate than the MNDVI, but is still in the range of other methods (see Table IX below). It can be seen that there is a registration problem between the sidewalk in the visual image and the TRF overlay. The authors believe this is due to the difference in viewing



a) Good vegetation detection with good false positive rejection.

b) Good false positive rejection, but less vegetation detection accuracy.

Figure 21. MNDVI and TRF Comparison O-D Cameras

Two sets of data each showing the original omnidirectional visual image, omnidirectional IR image, the unwrapped O-D visual and O-D IR images with the MNDVI vegetation detection with the Thermal Region Fusion vegetation detection results overlaid. Note that in a) TRF fused image shows a good capture of the vegetation and rejection of false positives. While in b) It has good FP rejection, but less accurate vegetation detection.

angle from the two cameras. The effects of this mismatch can be seen in Fig. 31 b).

3.3.Technical Approach for Vegetation Detection through Deep Learning Sensor Fusion of Omnidirectional (O-D) Far-infrared and O-D Visual Stream

Section 3.3. introduces the proposed system approach, which consists of the utilization of a CNN semantic feature extraction of the two streams coupled with deep learning to fuse the signatures from both the O-D vision and O-D Infrared cameras, and the application of a high-level fusion using DeepFuseNet. Two fusion networks will be applied to the individual sensor feature extraction, first using two Autoencoders feeding a CNN to output the vegetation features such as texture and color, and second, two CNN bottleneck feature extractors feeding the CNN-DCN Fusion Network (DeepFuseNet).

3.3.1. Overview of Technical approach for Deep Sensor Fusion

The proposed system architecture is to apply texture and frequency analysis (segmentation and signature) to images from low cost, O-D visual and IR sensors to provide vegetation detection, and classification. The final approach will be the fusion of visible and infrared perception systems using O-D sensors. Each block in the Fig 23. diagram represents the four subsections as follows:

- Section 3.3.2 discusses the O-D Camera System and the O-D Coordinate System setting.
- Section 3.3.3 Visual and Infrared Index, Histogram, and Segmentation Based Vegetation Detection

Table 15. Deep Fusion Network for o-d ir and visual stream

STEPS	STEP DETAILS

STEP 1	Apply the M NDVI vegetation index to extract vegetation regions.
STEP 2	$MNDVI = (I_{FIR} - I_{RED}) / (I_{FIR} + I_{RED})$
STEP 3	Apply thresholding to the IR image to identify regions with similar temperature within the vegetation band. $\sigma_{within}(T) = CDF_B(T)\sigma_B^2(T) + CDF_F(T)\sigma_F^2(T)$
STEP 4	Fuse results of Step 1 and 2 (Thermal Region Fusion)
STEP 5	Apply transfer learning with ImageNet and DL architectures along with Autoencoder Feature Extractors or Bottleneck Feature Extractors to find best network and parameters.
STEP 6	Apply DeepFuseNet architecture to fuse the O-D IR and Visual streams using the best architecture fed into a fully connected (Dense) CNN and pixel region classifier.
STEP 7	Compare Step 4 and Step 6 results

- Section 3.3.4 discusses the Autoencoder - feature extraction of the O-D IR thermal regions with the O-D Visual (Electro-optical) Multi-spectral signature for the regions of interest from the sensor streams.
- Section 3.3.5 discusses the Convolutional Neural Network (CNN) and Deep Learning (DL) tradeoffs and fine tuning.
- Section 3.3.6 discusses our DeepFuseNet architecture and the application of deep learning, transfer learning, and fine-tuning to solving the vegetation detection problem. Section 3.3.6 develops the computational foundation for a new approach to multi-spectral sensor fusion that identifies semantically significant object classes based on deep learning of the spectral and thermal signatures using DeepFuseNet to fuse the visual and IR spectrum components., finally.

The overall process is shown in Table 15.

The fusion of visual and IR O-D data will be used to enable the robotic perception system to adapt to different lighting and environmental conditions. In this work we will focus on two deep learning approaches and will compare them to the NDVI plus region segmentation fusion approach, the Autoencoder Feature Extraction, and the CNN fusion of O-D Far Infrared and visual stream with the System approach shown in Fig 23.

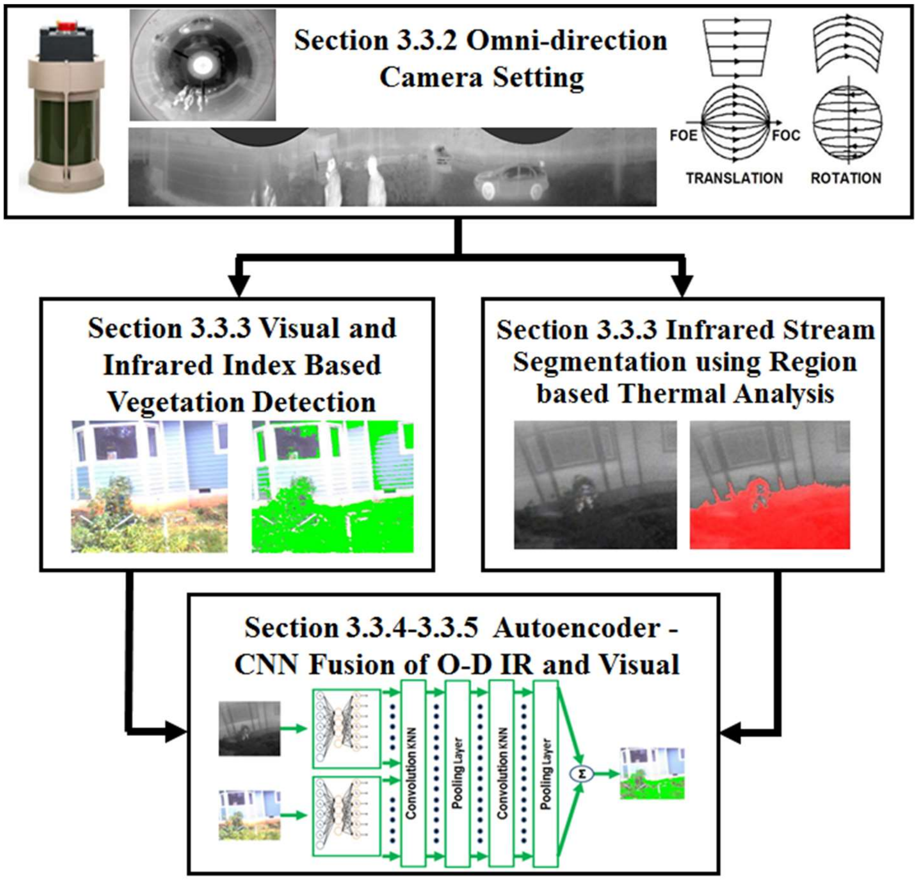

Section 3.3.2 Omni-direction Camera Setting
FOE
FOC
TRANSLATION
ROTATION
Section 3.3.3 Visual and Infrared Index Based Vegetation Detection
Section 3.3.3 Infrared Stream Segmentation using Region based Thermal Analysis
Section 3.3.4-3.3.5 Autoencoder - CNN Fusion of O-D IR and Visual
Convolution KNN
Pooling Layer
Convolution KNN
Pooling Layer
Σ

Figure 22. Deep Fusion Network Block Diagram
Technical Approach consist of the following subsections: III.A O-D Camera Setting, III.B) Visual and Infrared Index Based Vegetation Detection, III.C) Infrared Stream Segmentation using Region based Thermal Analysis, and III.D) Autoencoder – CNN Fusion of O-D IR and Visual Stream

3.3.2. Omni-direction Camera Setting

In this work we utilize spherical reflecting surfaces to achieve a projected 360-degree field of view. With these improved O-D systems, the 360-degree image is produced directly and does not require the fusion of multi-camera images. The transformational relationship between rectangular coordinates and spherical coordinates as shown in Section 3.1.1 Fig. 6 and described in equations (1-4).

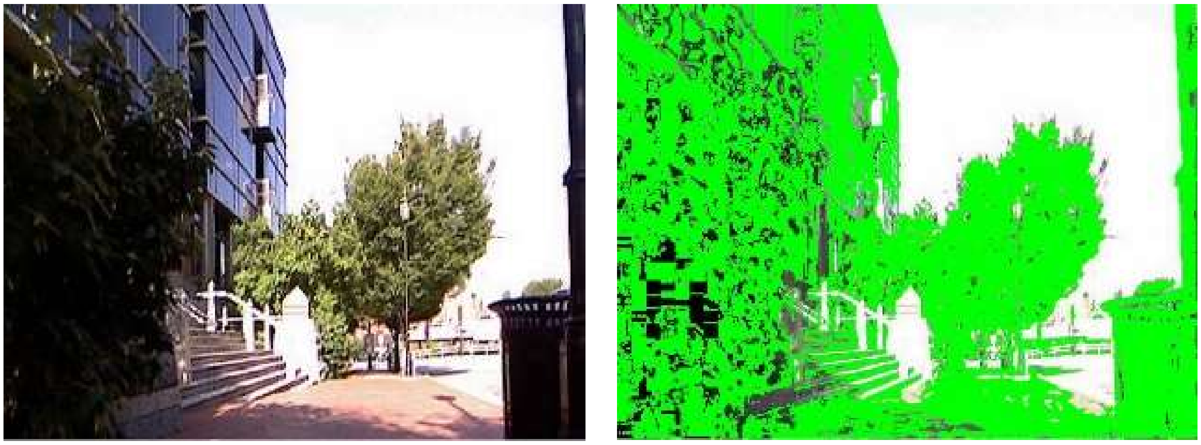
3.3.3. Visual and Infrared Index, Histogram, and Segmentation Based Vegetation Detection

In [44] we evaluated the effectiveness of two approaches, 1) the classical index based Normalized Difference Vegetation Index (NDVI) approach to vegetation detection which looks at the red color bands in the visual spectrum and compares this to near IR spectrum. The NDVI approach has a relatively high rate of false positives, and 2) a sensor fusion approach merging the NDVI and a thermal region segmentation approach. The NDVI feature works well in Chlorophyll rich vegetation, but doesn't do as well in dry vegetation or desert scenes. Our approach will use the DeepFuseNet deep learning approach described in Section 4.3.3 for sensor fusion and vegetation detection, and to improve performance in dry vegetation and false detects. We will compare this with the color vision MNDVI-region segmentation approach we used in [135].

The modified NDVI with thermal regions segmentation looks at the red band and far IR (FIR) band difference and fuses this with the thermally segmented region characteristic of the vegetation. Vegetation is detected based on the ratio of red reflectance and FIR reflectance fused with the limited thermal region of the vegetation. These techniques apply the band ratio of the FIR bands and red band in the visual range. Revisiting section 3.1, we use the Normalized Difference

Vegetation Index (NDVI), where $NDVI = (I_{NIR} - I_{RED}) / (I_{NIR} + I_{RED})$ modified for far-infrared (15) $MNDVI = (I_{IR} - I_{RED}) / (I_{IR} + I_{RED})$ We are using low cost O-D IR and O-D visual sensors to achieve better fused vegetation classification. Fig. 16 section 3.2.2 shows the location of a point of interest in the O-D IR and O-D visual sensor. The unwrapped IR image and the extracted relevant section from the IR and visual images are also presented. The MNDVI and TRF methods will be compared against our deep learning fusion approach called DeepFuseNet.

There are issues with false detects in the areas where the bands overlap when using just the MNDVI. We used the MNDVI approach on our data in Fig. 24 below to show the effect of this approach and a relatively large number of false detects.



a) Original Image *b) MNDVI Overlay Image*

Figure 23. MNDVI False Detects
MNDVI Approach: (a) Original image, (b) Processed image using MNDVI vegetation index approach. This result has a high number of false detects.

We as humans are accustomed to using color and texture to distinguish materials. The color and reflectivity are important indicators of the material composition of an object. The approach we demonstrated in [132] was to fuse the MNDVI (5) and thermal IR region (22),

$$\sigma_{withi}\ (T) = CDF_B(T)\sigma_B^2(T) + \ CDF_F(T)\sigma_F^2(T) \tag{22}$$

where, CDF_B (T) is the cumulative density function of the background below threshold T, $\sigma_B^2(T)$ is the variance of the background histogram at T, and CDF_F (T) is the cumulative density function of the foreground above the threshold (T), and $\sigma_F^2(T)$ is the variance of the background histogram at T. We then utilize the fused signatures to extract scene semantics from the O-D IR and visual images. We demonstrated that each material has a characteristic visual and thermal signature from which to classify the material. In this work, we compare the results from the MNDVI red band approach and the fused IR region based thermal segmentation approach, and finally compare these two methods to two deep learning approaches presented in Section 4.3.3.

In [132] we address the issue of false detects in the index based approach, using semantic extraction based on thermal regions to accentuate vegetation detection and reduce the number of false positives. The thermal image has similar materials clustered in the same region of the infrared color space. At different ambient temperatures these materials will shift in color but will remain clustered. By using deep learning to recognize and learn this region based color signature and to also learn the thermal behavior we can merge the features learned by the two networks. During the day vegetation is cooler than the soil and is darker in the IR image. At night vegetation and soil reverse and the vegetation is lighter. They are also visible in the IR image when the visual is obscured by dust or fog. These attributes are an advantage of applying deep learning to both the visual and the thermal information.

3.3.4. Autoencoder - Convolution Neural Network

The contribution of this work is a computational foundation for a new approach to multi-spectral sensor fusion, that identifies semantically significant object classes using sensor fusion of

O-D Vision (Electro-optical) and O-D IR sensors applying the two streams into two Sparse Autoencoder Feature Extractors (SAFE) with a CNN back end for the fusion of the IR and Visual Spectral content to extract the regions of vegetation and other materials. Our algorithm will apply several Sparse Autoencoder Feature Extractors into the input of the CNN for fusion / classification. We will evaluate the SAFE-CNN architecture against the baseline index based vegetation detection.

Our algorithm will apply several multilayer kernel filters into the input of the Fusion CNN (DeepFuseNet). We will evaluate two deep learning architectures against the baseline index and thermal region based fusion, which are,

- An Autoencoder / Convolution Neural Network
- DeepFuseNet – two Conv Neural Networks which apply transfer learning merged into a final output network.

3.3.4.1. Auto-encoder Neural Network

The first architecture will feed the two visual and IR input streams into two respective Autoencoder networks that will feed forward the input image and minimize the difference between the input and output to learn the features of the materials in the scene. These two Autoencoder outputs will then be fed into a CNN to fuse their features and provide a fused sensor output.

Two kernel filters will be applied:

- MNDVI based vegetation detection kernel pre- filter
- Region based segmentation kernel pre-filter

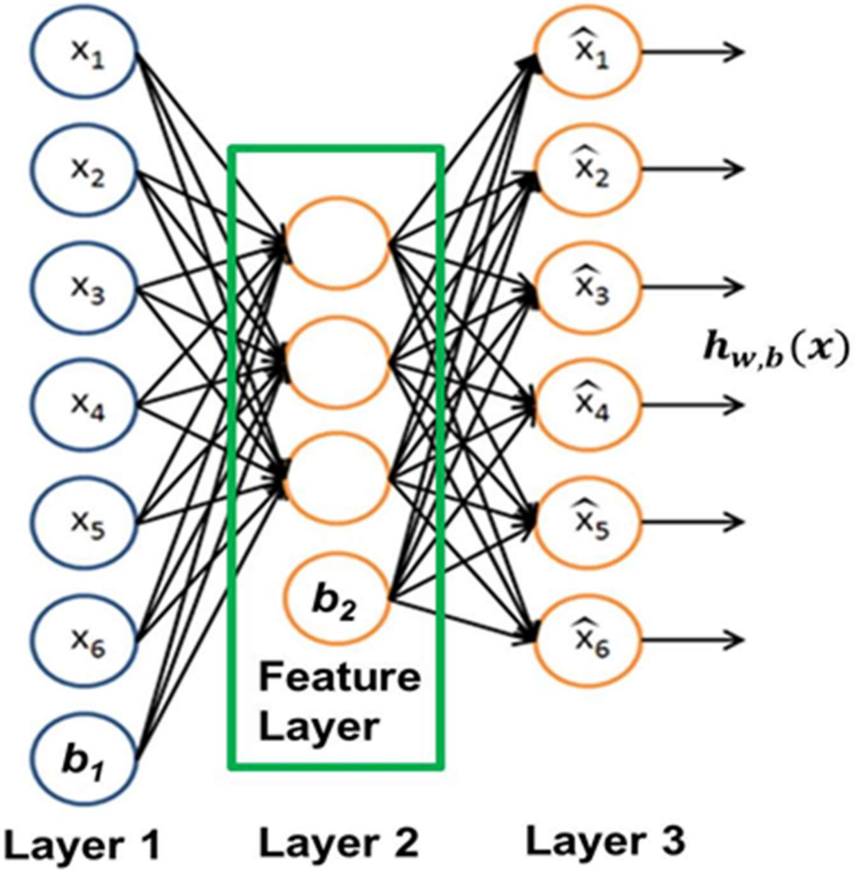


Figure 24. Autoencoder Layers

Shows a three-layer Neural Network with a hidden layer that is sparse or constrained which forces the network to learn certain features in making the output match the input. This is an Autoencoder.

The Sparse Autoencoder architecture Fig. 25 will feed the two visual and IR input streams into two respective Sparse Autoencoder networks that will feed forward the input image and minimize the difference between the input and output to learn the features of the materials in the scene. These two Autoencoder outputs will then be fed into a CNN to fuse their features and provide a fused sensor output. The overall process was given in Table 22 in Section 4.3.

The Autoencoder will be built up of a network of basic "Neuron" nodes (Fig. 25) and will apply unsupervised learning that will approximate the kernel feature map of the scene.

3.3.4.2. Sparse Auto-encoder

In our approach we will use an unsupervised learning algorithm where the neural network will be an Auto-encoder (Fig. 25). An Auto-encoder is a Neural Network that applies backpropagation in unsupervised learning where it is trying to match the output to the input. We start with a set of unlabeled training data $\{x1, x2, x3, xn\}$ where $x_i \mathcal{E} R^n$ and the network learns by matching the output to the input, or by setting $y_i = x_i$. As the weights and biases and activations are set to minimize the error between the two, the hidden/ feature layers will learn patterns in the image set. The Auto-encoder will try to learn the output function h, such that $h_{Wb}(x) \approx x'$ is satisfied to make the output match the input, which means that it is trying to approximate the identity function.

By putting constraints on the Auto-encoder Neural Network the Auto-encoder will learn some of the structure in the data. For instance, if some of the features are correlated the Auto-encoder will detect this pattern. There are several ways to put constraints on the network. One is to limit the number of units in the hidden layer. Another is to place a sparsity constraint on the hidden units in the network. With a sparsity constraint the Auto-encoder will still be able to detect interesting structures in the data even if the number of hidden units is large. If we use a sigmoid function (23),

$$a_i^l = \frac{1}{(1+e^{-x})} \tag{23}$$

as our activation parameter we can think of a given feature as being active if its activation for feature i at layer l, a_i^l is close to 1 and inactive if it is close to0. We want each feature node in the hidden layer to be inactive most of the time. We want that location to be activate only when there is sufficient correlation in the data. Therefore, we will put a constraint on the cost function that drives this condition. This sparsity constraint parameter will be added to our cost function to place a penalty on a neuron activating if the image is not correlated in the region of interest. This will be the case if the feature is not present. The activation a_i^l represents the activation of the hidden node at the l^{th} layer for a given input *x (i)*, as shown in (24).

$$\widehat{p_J} = \frac{1}{m}\sum_{i=1}^{m} a_j^l \, x^{(i)} \tag{24}$$

We will then enforce the constraint that $\widehat{p_J} = \rho$ where ρ is the sparsity parameter and we will constrain it to be small and close to zero, such that the outputs match the input within the convergence error. Our Cost function will then be (25)

$$J(W, b; x, y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2 \tag{25}$$

We will apply the Kullback-Liebler *(KL)* divergence (26) constraint to penalize $\widehat{p_J}$ the farther it diverges from zero.

$$\sum_{j=1}^{n_l} KL(\rho\,||\widehat{\rho_J}) = \sum_{j=1}^{n_l} \rho \, log\frac{\rho}{\widehat{\rho_J}} + (1-\rho)\, log\frac{(1-\rho)}{(1-\widehat{\rho_J})} \tag{26}$$

This will force $\widehat{p_J} \approx \rho$ and our cost function becomes (27).

$$J_{sparse}(W, b) = J(W, b) + \beta\sum_{j=1}^{n_l} KL(\rho\,||\hat{\rho}) \tag{27}$$

To learn the weights for the nodes in the network we will conduct a forward pass with initial weights, and then use backpropagation to refine the weights to match the output to the two input streams from the O-D infrared and visual cameras. Backpropagation will be used to adjust the weights as follows (28).

$$\delta_j^l = \left((\textstyle\sum_{j=1}^{n_l} W_{ji}^l \, \delta_j^{l+1}) + \beta\left(-\frac{\rho}{\hat{\rho}} + \frac{1-\rho}{1-\hat{\rho}}\right)\right) f(\acute{z}_i^l) \tag{28}$$

Because of the sparse constraint on the network, as the network converges to the optimum it learns the new weighting on the hidden layers to match the output to the input. Since it is constrained, it will find the features in the data and key structures in the two input streams will

emerge. These key structures will then be fed into a pre-trained CNN to fuse the features. This architecture is shown in Fig. 26.

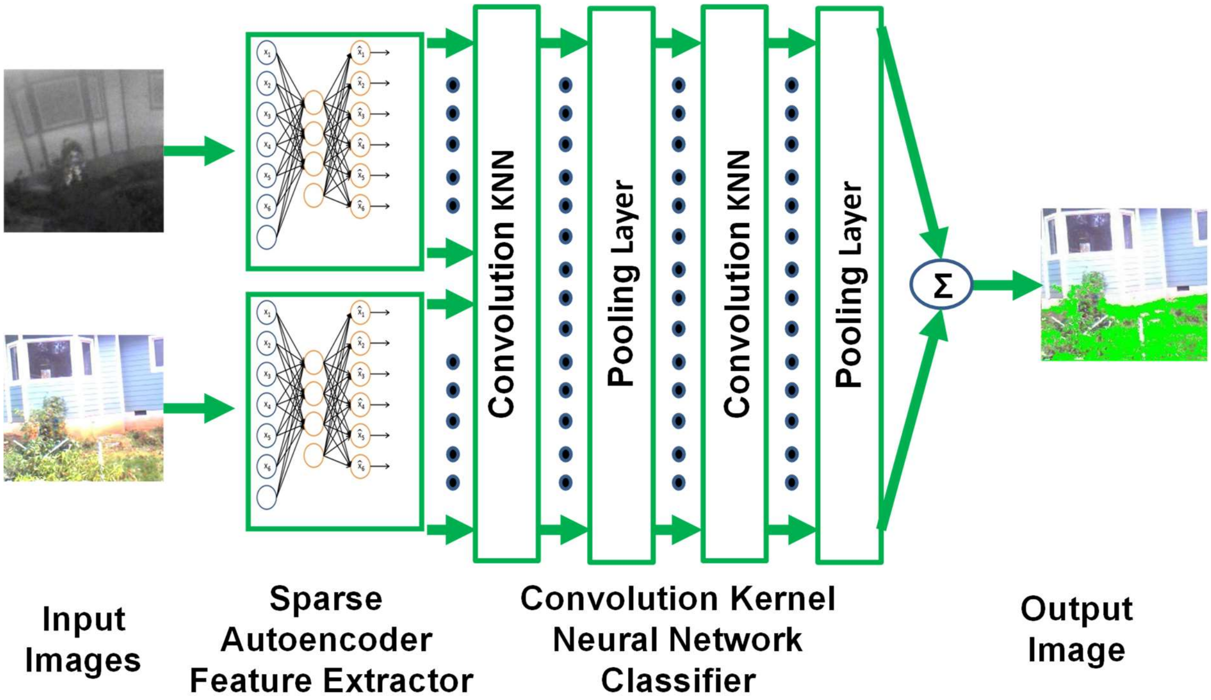


Figure 25. Autoencoder Fusion

Shows the first fusion architecture with two Autoencoders feeding into a two-layer CNN with max pooling layers.

3.3.5. Convolution Neural Network (CNN)

The CNN will consist of two conv kernel layers. Each conv layer will be followed by a Rectified Linear Non-Linearity (ReLU) layer, and then a max pooling layer.

At each conv layer the feature map from the previous layer is convolved with the learnable kernel two filter for the current layer and input into the activation function to form an output feature map and the i^{th} feature map in layer l Y_i^l, is denoted by (29).

$$Y_j^l = f(B_j^l + \sum_{i=0}^{m_1^{l-1}} Y_i^{l-1} * K_{ij}^l) \tag{29}$$

Where the output at layer l, Y_j^l, is a function of the matrix B_j^l which is the matrix of bias at each layer l added to the sum of the spatial conv of the output of layer l-1, Y_i^{l-1}) with the matrix of Kernel, K_{ij}^l at layer l over an i x j window

The size of the output feature map is given by (30).

$$M_x^l = \frac{M_x^{l-1} - K_x^l}{S_x^l + 1} + 1; \quad M_y^l = \frac{M_y^{l-1} - K_y^l}{S_y^l + 1} + 1 \tag{30}$$

Where each layer has M maps of equal size (M_x, M_y). The kernel filter is shifted over the image such that it does not go outside the image. The kernel is of size (K_x, K_y), where the index l indicates the layer number, and each map in layer L^l is connected to a subset of maps in the previous layer $L^{(l-1)}$.

The max pooling layer is a down-sampling layer that reduces the dimensionality of the image. It also improves the spatial invariance by reducing the resolution of the image. The pooling window is a *k x k* window. In our case we will take the maximum value and apply it to the *k x k* kernel patch utilizing (31).

$$m_j = \max_{KxK}(m_i^{kxk} u(k,k)) \tag{31}$$

where, m_j is the max value that each pixel in the max pooling window is set to after the search of the window u $(k \times k)$ for the max value. Each of the kernel layers will represent a refinement of the image representation with each layer being more invariant than the previous. These kernel layers will represent the feature maps of the image.

Each of these kernel image feature maps will have normalized coordinates both in the image and the Hilbert Space (H). The higher dimension kernel feature map is the dot product of close layer features. Applying the multilayer convolutional kernel to the image feature map will result in patches of like features and the fusion of the kernel pre-filters. This represents a spatial conv over the image. The resulting semantic feature vector will be the input to the 2D vegetation classification algorithm.

The following parameters characterize each conv layer: size and number of maps, the kernel window size, the stride (step over or skipping factor), and the connection strategy. The conv layers of our network have the following key parameters in line with the findings of [27].

Stride – The spacing between patches where features will be extracted or in other words the number of pixels skipped before again applying the receptive field. The results showed the best performance at $S=1$ with a clear downward trend in performance as step size increases. This is a trade-off between accuracy and computation time.

Kernel Size – the Kernel Size is the window or field size over which the features are extracted. [27] used 6, 8, and 12 pixels. Generally, 6 pixels worked best. We will use 2, 3, and 6 pixels

Number of Features - [27] evaluated 100, 200, 400, 800, 1200, and 1600 leaned features. On average, the algorithms performed better by learning more features, although the increase above 800 was gradual.

We did a study looking at the performance of various model architectures and parameters. We evaluated the following architectures and parameters. Architectures evaluated were AlexNet, SqueezeNet, ResNet50, VGG16, ResNet101, ResNet152. Parameters evaluated were Learning Rate(LR) η, and Momentum *m.* Learning Rate η is the rate at which the weights are updated the weight update process is shown in (32).

$$w \xleftarrow{update} w - \eta \frac{\partial L(x,y)}{\partial w} \tag{32}$$

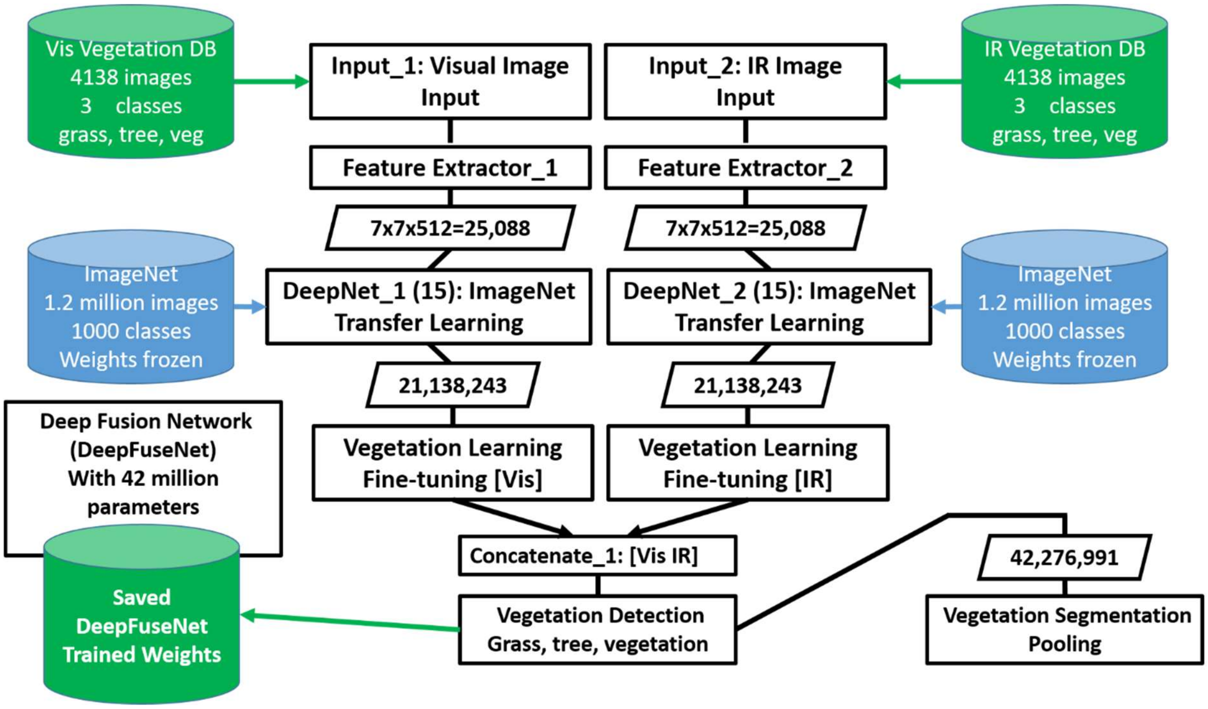
We evaluated LR of 1e-1, 1e-2, 1e-3, 1e-4, and 1e-5. The learning rates were adjusted to minimize overfitting. Where $\frac{\partial L(x,y)}{\partial w}$ is the backpropagation method of updating the weights in the network by comparing the actual output to the desired output with the partial derivative of the error with respect to the weights, and *L(x,y)* is the cross-entropy loss function and is given by (33)

$$L(x,y) = -[y\, lo \qquad + (1-y)\, log(1-x)] \tag{33}$$

Momentum *m* – we evaluated momentum values of 0.9, 0.8, 0.75, 0.7 utilizing the above architectures trained on ImageNet. Dropout *d* was applied to reduce overfitting by randomly dropping out neurons in the network at probability *p.*

3.3.6. Deep Learning Fusion Network (DeepFuseNet)

Our architecture (functional depiction Fig. 27), for the deep learning fusion network (DeepFuseNet) will use two feature extractors fed into two deep learning VGG 16-layer deep learning convolution networks trained on ImageNet, one for the visual and IR streams. We then apply transfer learning by freezing the lower 15 layers and training a new fully connected top layer, the new head model, on our smaller data set. Each layer will perform conv followed by ReLU non-linear layer and a max pooling layer with a 2x2 window and down-sample. We conducted a series of experiments to fine tune the parameters of the new top of the DeepFuseNet vegetation learning network. We feed either two Autoencoder feature extracting networks or two bottleneck feature extracting networks into the two learning networks before concatenating into the final top two fusion network and classifier layers. Fig. 27 presents the functional architecture for the two parallel VGG 16 DeepNet models with five convolutional / max pooling blocks and the concatenated input into the final vegetation detection classification layers. The output is then fed into a vegetation region pooling block. The final step is to generate the vegetation mask over the image. Fig 6 in chapter 1 section 7 shows a high level depiction of the deep learning training and deployment setup. Each leg of the network generates 21.1 million parameters of which 14 million are trainable and the rest are frozen with imagenet weights and are not trainable. The total DeepFuseNet merged model generates 42 million parameters. The saved weights are then fed into the deployed system so they do not have to be relearned.



Vis Vegetation DB
4138 images
3 classes
grass, tree, veg
Input_1: Visual Image Input
Input_2: IR Image Input
IR Vegetation DB
4138 images
3 classes
grass, tree, veg
Feature Extractor_1
Feature Extractor_2
7x7x512=25,088
7x7x512=25,088
ImageNet
1.2 million images
1000 classes
Weights frozen
DeepNet_1 (15): ImageNet Transfer Learning
DeepNet_2 (15): ImageNet Transfer Learning
ImageNet
1.2 million images
1000 classes
Weights frozen
21,138,243
21,138,243
Deep Fusion Network (DeepFuseNet) With 42 million parameters
Vegetation Learning Fine-tuning [Vis]
Vegetation Learning Fine-tuning [IR]
Concatenate_1: [Vis IR]
42,276,991
Saved DeepFuseNet Trained Weights
Vegetation Detection Grass, tree, vegetation
Vegetation Segmentation Pooling

Figure 26. Merged Model Functional Block Diagram

Merged Model functional block diagram with two VGG16 Deep Networks with two inputs, one from visual camera and one from IR camera.

The two concatenated networks are fed into a top network of two dense conv layers and a softmax classifiers. Fig 28 represents the concept depiction of the DeepFuseNet. Fig. 28 shows the visual and IR image each being fed into a feature extractor and then feeding the DeepFuseNet. The output of the network is then fed into a softmax classifier to output the image with the vegetation areas detected and classified.

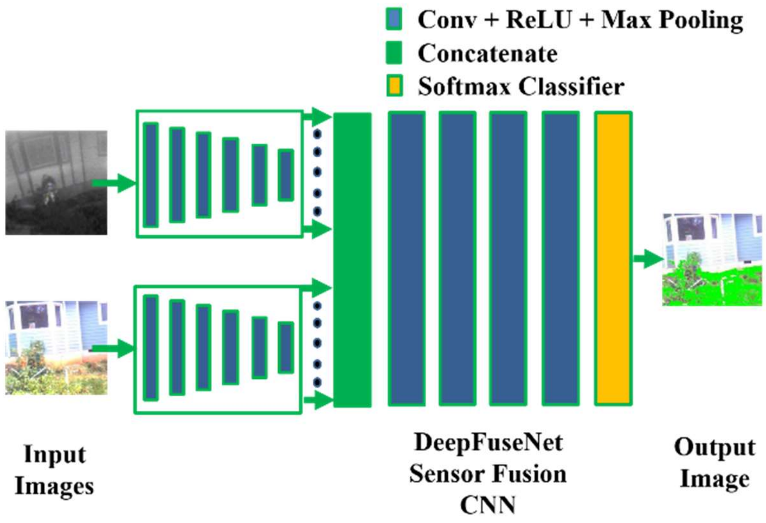


Figure 27. Deep Fusion Network (DeepFuseNet)

Shows the DeepFuseNet fusion architecture with two CNN feature extractors feeding into a seven-layer CNN with max pooling and a classification layer

3.3.7. Validation Methodologies

The validation approaches typically used are generating a train/validation/test split from the dataset. Typically, the train/validation/test split is 60%/20%/20% respectively. To improve the robustness of the data set the training, validation and test sets can have random augmentation methods applied. The augmentation approach randomly picks images and changes them in some way. Several methods are to rotate the image by some angle, or flip the image.

Advantages of train/val/test split**:**

- Runs K times faster than K-fold cross-validation
    - This is because K-fold cross-validation repeats the train/test split K-times
- Simpler to examine the detailed results of the testing process

Another approach to validating the results for a small data set is to do K-fold cross-validation. This approach takes the entire data set and does a moving validation and then averages the K-fold ensamble results. Say K = 5, the data set is split into five smaller data sets each 20% of the data. For the first run, K_1 is the validation data set and K_2-K_5 are the training set. At the end of the training the results are validated against K_1. The next run a different K_i is used for validation and the remaining for training until all combinations have been run. Then the five validation ensambles are averaged to get the overall validation result.

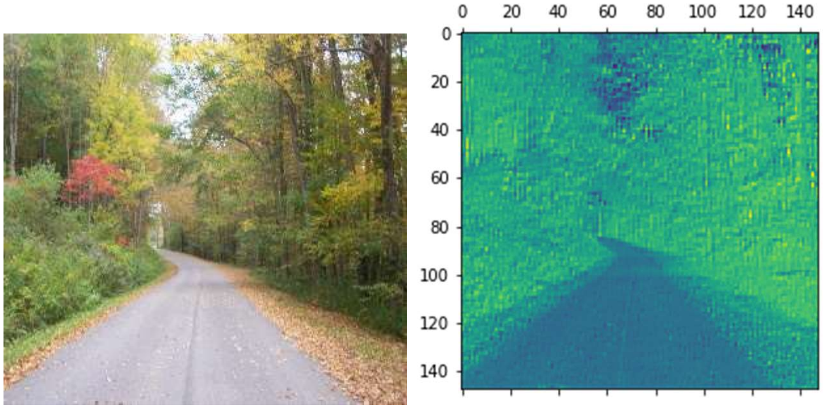
Advantages of K-fold cross-validation:

- More accurate estimate of out-of-sample accuracy
- More "efficient" use of data
  - This is because every observation is used for both training and testing

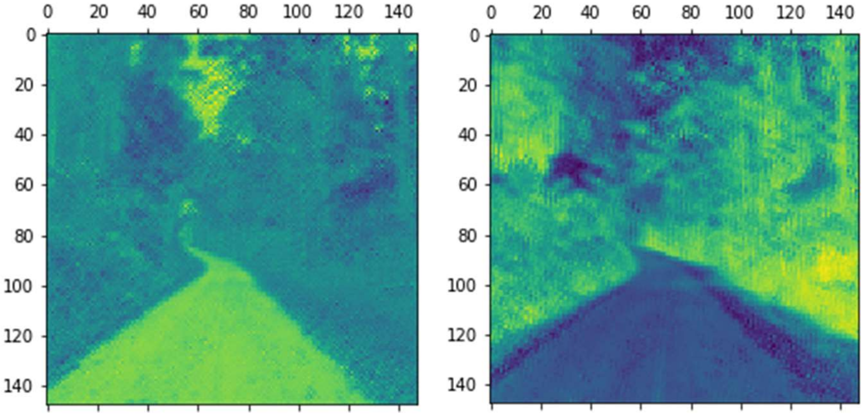
We used the standard train/validation/test split for our data.

3.3.8. Visulization of Network Filters

We utilized network filter visualization techniques [41] to visualize what features were activating the network activation filters. Fig. 29 a) is the original image, b) channel 3 is activating on vertical edges, c) channel 13 is activating on the sky and road surface, d) is activating on vegetation texture. The conv filter maps for the first conv layer and three representative channels are presented in Fig 29. In Fig. 29.b) the filter is learning vertical edges in the vegetation. In Fig. 29.c) the filter is learning primarily the sky and road surface which are both a gray color in this example. Finally, in Fig. 29.d) the filter is learning tree trunks and splotches of leaves.



a) Original Image *b) Conv1-3 feature map*

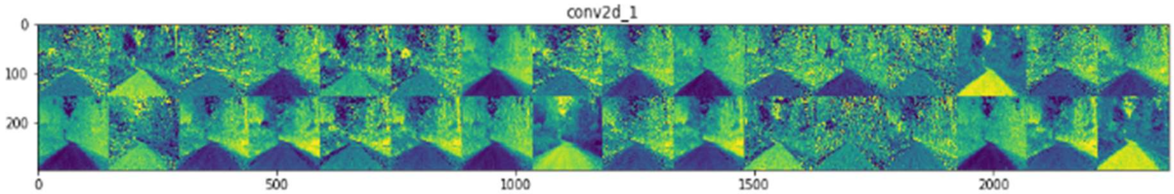


c) Conv1-13 feature map *d) Conv1-19 feature map*

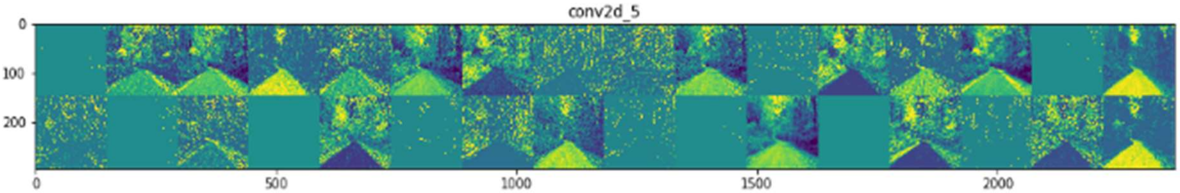
Figure 28. Convolution Layer Feature Map Visualization

First conv layer feature maps for three out of the 32 channels. a) Original image, b) Feature map for layer 1 channel 3, c) Feature map for layer 1 channel 13, and d) Feature map for layer 1 channel 19

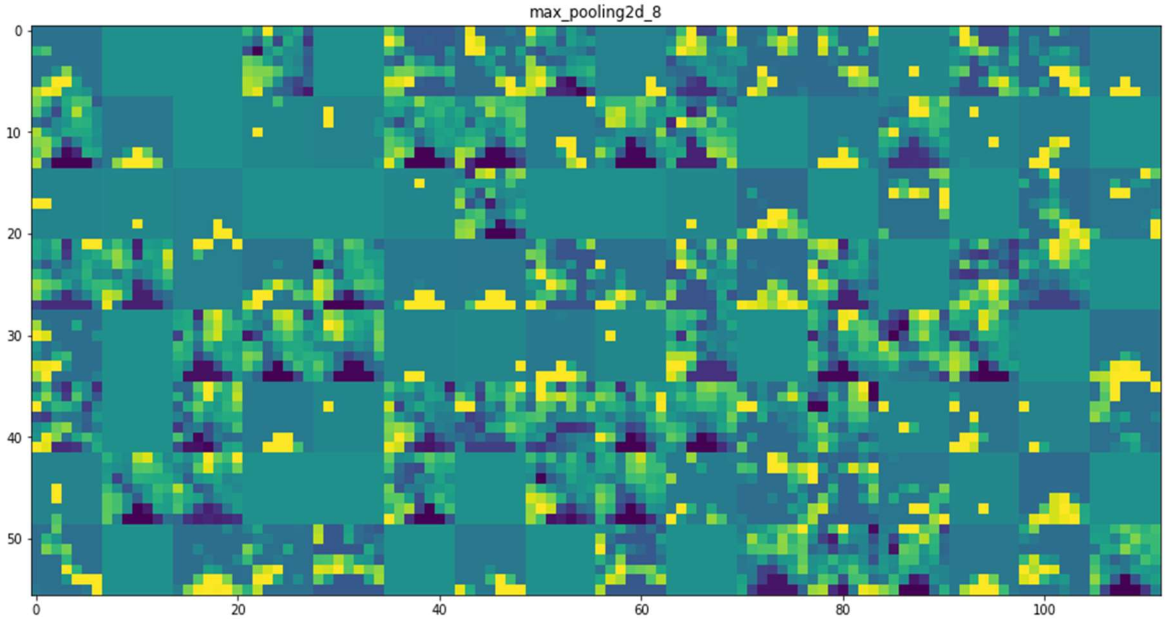
Fig. 30 shows the activation filter maps for several layers in the network. Fig. 30.a) are the 32 filters for convolution layer 1, Fig. 30.b) are the 32 filters for convolution layer 5, and Fig. 30.c) are the 128 filters for the 8th Max Pooling layer. It can be seen from Fig. 29 and 30 that the various activation filters are learning(activating) on different feature sets. In Fig 30 the various filters are showing that certain features are beginning to dominate.



a) Conv Layer 1



b) Conv Layer 5



c) Max Pooling 8

Figure 29. Activation Filter Map Visualization

Activation filters for two convolution layers and a max pooling layer.

4. Chapter – Experimental Results for the Three Methods

4.1.Direct Spherical Calibration Experimental Results

Section 4.1 describes the data setting for the evaluation. Section 4.1.1 applies the DSC methodology presented in Section 3.1, Section 4.1.2 compares DSC to the three Baseline methodologies, Section 4.1.3 presents the error analysis of four methods, and 4.1.4 presents the results of the optimum number for capture of calibration boards.

4.1.1. Calibration data setting for evaluation

In the use of the omnidirectional IR camera, a camera calibration process is required just as in a conventional vision system. The O-D camera operational setting is shown in Fig. 31 and is an example of an omnidirectional IR camera mounted on our Pioneer robot; the figure shows a) the camera on the robot, b) a sketch of the internal mirror, and c) a representative IR output. A car, a truck, and three people can be seen in the image. This work is laying out the calibration approach and initial experimental results using data gathered with our camera mounted on the robot. The resulting images were visually inspected to determine if the quality and clarity of the images were sufficient for the calibration work. The images were also evaluated to determine if the vertical support bars of the camera significantly obscured the view of the calibration boards.

Table 16. Calibration Image Sets

Image set	# Image	Resolution	Obscured	Clarity
Sparse Set	10	61%	Four partial	4 of 10 clear
Moderate Set	25	70.4%	Five Partial	16 of 25 clear
Dense Set	47	53%	Fourteen Partial	28 of 47 clear

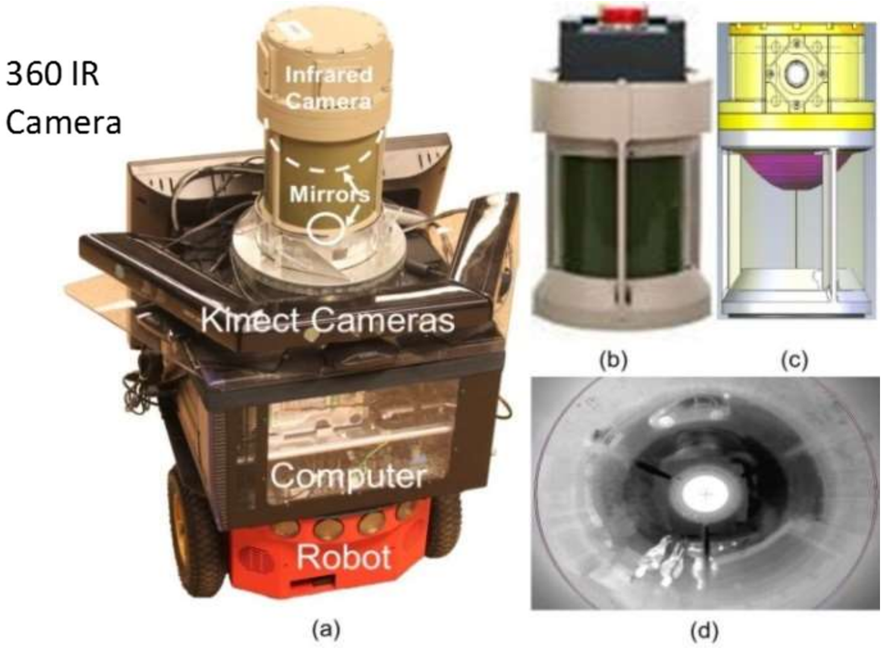


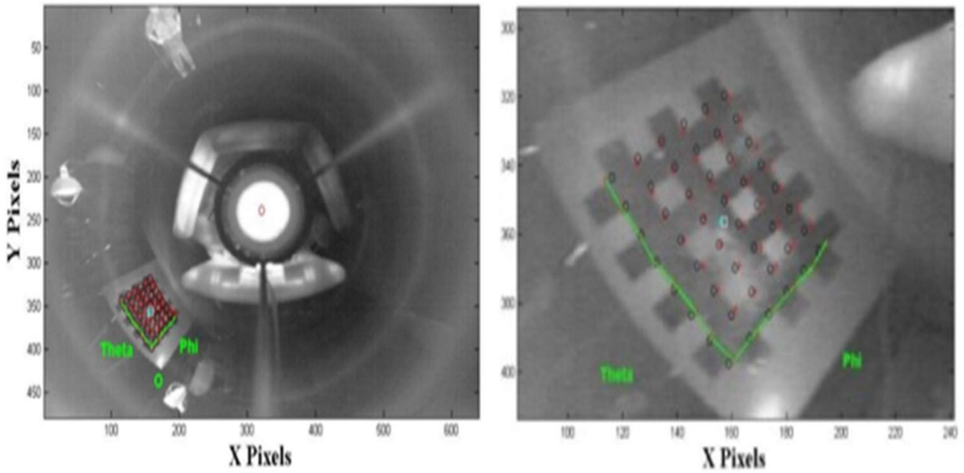
Figure 30. Robot and Camera Setup:
Robot with Omnidirectional IR camera showing a) the Robot, b) the IR camera, c) the camera solid model showing the hemispherical mirror, and d) a representative output showing 3 people, a truck, and a car.

The three data sets were taken in separate setups. The sparse set had fair quality, some partial

Table 16. describes the characteristics of the three calibration data sets used for the evaluation.

obscuration from the support bars on the camera, and moderate clarity. The moderate data sets had good quality and good clarity.

The dense data set had marginal quality. The moderate set had no images obscured while the dense had the most partial obscuration with ten out of forty-seven of the calibration board views. These rankings were subjectively determined by visual observation of the images.



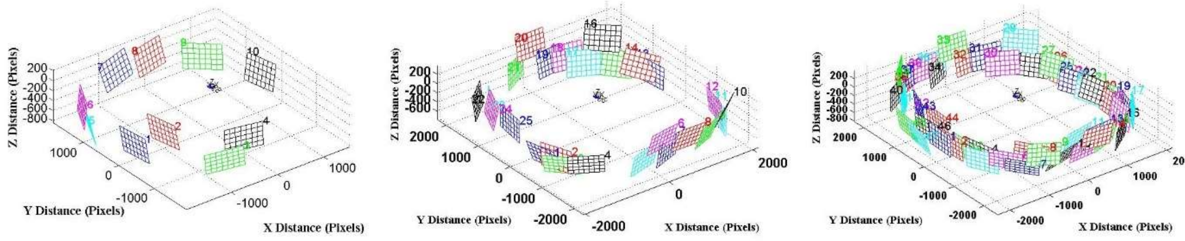
a) Re-projection on grid *b) Zoomed in calibration grid*

Figure 31. Calibration Re-projection points:

Example of re-projection points in the Baseline 1 (distortion) method with a) Grid points chosen for calibration and re-projection of points onto the calibration pattern. b) Zoomed in view of the calibration grid.

Fig. 32 shows a sample of calibration points picked by the operator using the Baseline 1 (distortion): Scaramuzza *et al*. methodology, and a typical re-projection of the calibration points onto the original pattern. We modified the plot to show the theta and phi directions on the plot. The heat lamps can be seen in the images as well as the reflected calibration grids. Utilizing the four methods; our DSC method, the Baseline 1 (distortion) method [10 - 12], the Baseline 2 (single viewpoint spherical) method [13], and the Baseline 3 (generic) method [41-42]; each calibration methodology was applied to the IR catadioptric camera using a calibration pattern of a white board with thermal reflective tape making a grid pattern that could be used to create a thermally reflective pattern on the board.

The checkerboard pattern was required by the three Baseline methods. As a result, the DSC method used the same checkerboard pattern that was used for the three Baseline methods for a more direct comparison. Even though our method only used the outside corners to find the center, we wanted to be able to compare to the results for the other methods directly without introducing additional factors. The images were then captured and the four methodologies applied. The experimental result comparisons are presented in Section 2.4.2 and 2.4.3. Fig. 33 is the data setting and shows the extrinsic projection of the three data set positions around the camera.



a) Sparse data set. *b) Moderate data set.* *c) Dense data set.*

Figure 32. Extrinsic Mapping:
Extrinsic mapping of the three data sets, a) sparse data set with 10 calibration images, b) moderate data set with 25 calibration images, c) dense data set with 47 calibration Images,

4.1.2. DSC calibration

Our DSC approach was adapted to more effectively work with an omnidirectional IR camera, and to improve the accuracy over the Baseline methods. The DSC methodology approach to the corner selection dramatically reduced the error introduced by the Baseline corner and blob selection approaches due to their algorithm failing when given the low resolution of the IR image, and Fig. 34 presents the comparison of the DSC to the baseline methods for the sparse (10 calibration images) data set.

The plots in Fig. 34 represent the average error of the four methodologies across the calibration boards. The error for the DSC method was the smallest and was on the order of 0.2 to 0.5 x 10-4 pixels for all of the sparse data set. Baseline 2 (single viewpoint spherical) was next best with 1-2 pixel error, Baseline 1 (distortion) with 2-3 pixel error, and finally Baseline 3 (generic) with error on the order of 8-19 pixels of error.

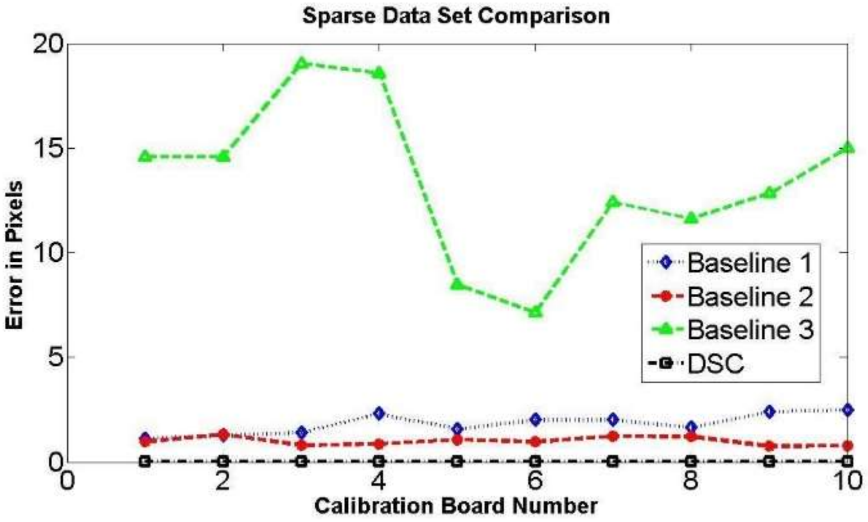


Figure 33. Sparse Data Set Error
Comparisons of the re-projection error for the three Baseline methods with our DSC method. The results are shown for the sparse data set case. The DSC method has the lowest error followed by Baseline 2 (spherical), then Baseline 1 (distortion) and finally Baseline 3 (generic) as the worst.

Note that Baseline 1 (distortion) and 3 (generic) both tended to go out of bounds due to poor geometry selection with the lower quality IR images and required an inordinate amount of hand adjustment to get them to work. Fig. 35 shows the comparison of the four methods for the moderate data set blurred IR images. The calibration images were captured and processed using our DSC methodology. The corner selection methodology was an initial manual selection with corner assist to find the entire capture board grid and then find the center of the grid. We used the direct spherical coordinates of this calculated center point for each calibration board grid as the

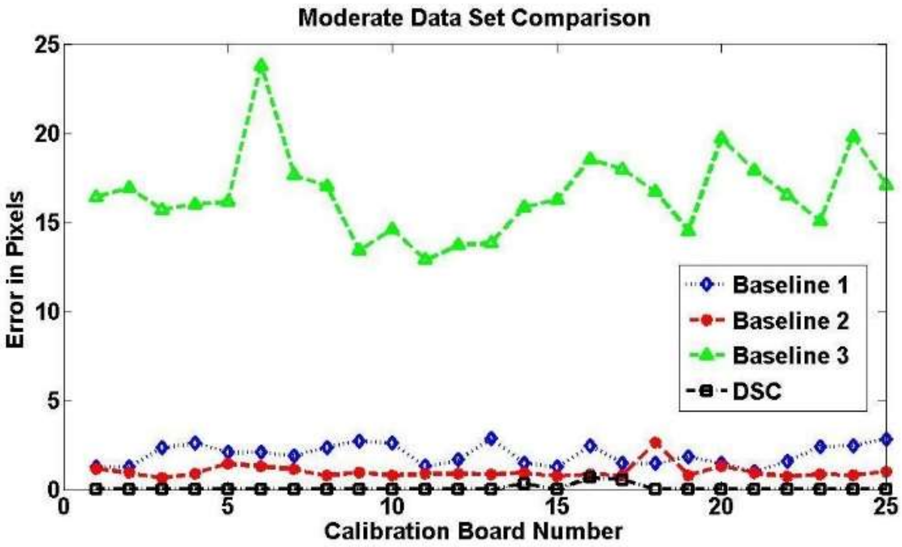


Figure 34. Moderate Data Set Error
Comparisons of the re-projection error for the three Baseline methods with our DSC method. The results are shown for the moderate data set case. The DSC method has the lowest error followed by Baseline 2 (single viewpoint spherical), then Baseline 1 (distortion) and finally Baseline 3 (generic) as the worst.

input to the DSC calibration. This worked much better for the IR images and significantly cut down the analysis time. This method was more robust to the poor edge quality of the IR image.

4.1.3. Comparison to other Baseline calibration methods

Among the existing color vision omnidirectional camera calibration methodologies, we chose the two best performing methods plus one more generic method for comparison.

The Baseline 1 (distortion) omnidirectional camera calibration methodology, [7], [8], [9] proposed a polynomial approximation distortion model with 2D patterns approach.

In the Baseline 2 (single viewpoint spherical) methodology [10], single view point re-projection method uses a distortion model of the camera properties with a single viewpoint spherical projection model where the world points are projected onto the unit sphere. They then use a secondary projection onto a normalized plane.

In the Baseline 3 (generic) methodology [34-35] was chosen as a more generic method intended for a wide range of camera types. It uses a generic model of the camera properties that could handle different types of cameras including conventional perspective, central omnidirectional (including catadioptric, fish-eye, and generic central lens), and finally non-central cameras. The approach utilizes two Fourier series polynomial distortion terms in the calibration process. The distortion model is more generic and not based on the perspective projection.

The three Baseline calibration methodologies were compared to our DSC methodology. As the data for the DSC shows the smallest error with the moderate data set having 22 out of 25 calibration boards with error less than 1 x 10-4 pixels with a couple of sub-pixel error outliers. The DSC moderate data set has three outliers at 0.34, 0.54, and 0.69 pixels.

Again the Baseline method 1 and 3 methods required many tries at getting the solution not to go out of bounds and this was very labor intensive since the automatic corner finding routine was optimized for visual omnidirectional images and wouldn't work with the lower quality IR images, this made the method 1(distortion) and 3 (generic) methods automatic selection of points unreliable. Additionally, the manual corner selection was error prone causing the routine to blow up and fail necessitating rework. This was quite time consuming and took about a week of 5-6 hours a day making manual corrections to the corner coordinates for Baseline 1 (distortion) and similar timeframe with Baseline 3 (generic) trying to get the solution to work. This still did not correctly find all the corners. This required us to go into the images and manually reposition about a third of the corners on about half of the images. After this correction the max and average errors behaved much better. Fig. 35 shows that the error comparison for the moderate data set was similar to the sparse case with DSC best, Baseline 2 (single viewpoint spherical) next, Baseline 1 (distortion) next with 2-3 pixel error, and finally Baseline 3 (generic) with 13-25 pixel error.

Fig. 36 shows the comparison of the methods for the dense data set. We were not able to show the Baseline 3 (generic) since after a week of attempting to get it working the program would crash when trying to find the calibration points in the low quality IR image.

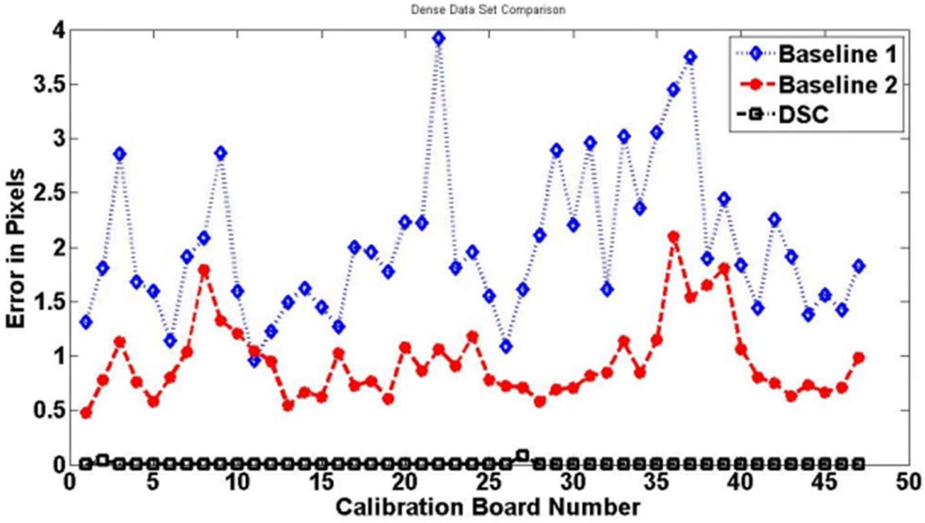


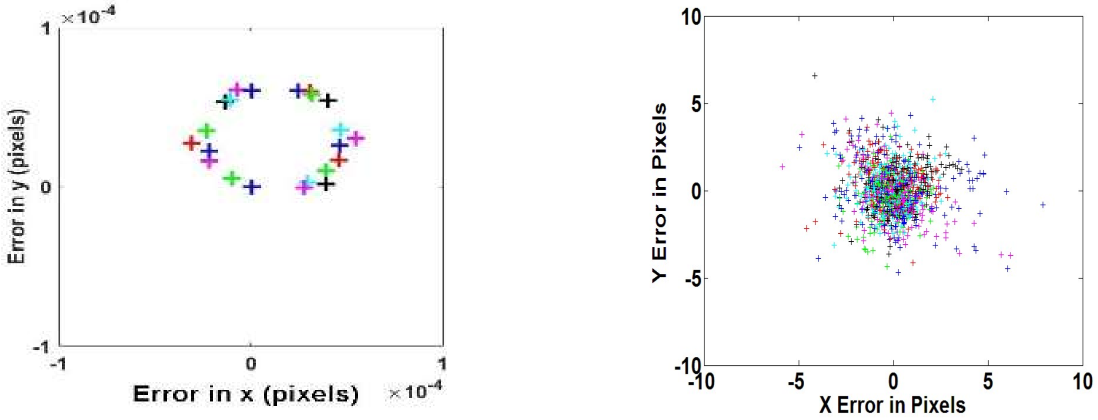
Figure 35. Dense Data Set Error

Comparisons of the re-projection error for the three Baseline methods with our DSC method. The results are shown for the dense data set case. The DSC method has the lowest error followed by Baseline 2 (spherical), then Baseline 1 (distortion) and finally Baseline 3 (generic)as the worst.

As before with the sparse and moderate data sets DSC is the best with sub pixel error with only two outliers near 0.1 pixel and the rest less than 10^{-5}. Baseline 2 (single viewpoint spherical) is next with 1-2 pixel error, and Baseline 1 (distortion) with 1-4 pixel error. Again Baseline 3 (generic) would blow up.

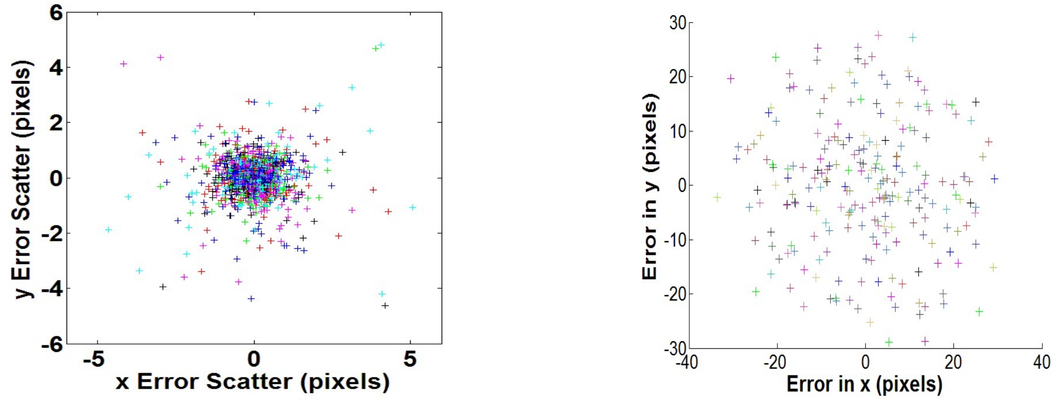
4.1.4. Error analysis of four methods

In Fig. 37 we compare the x, y error scatter for the four methodologies. a) DSC error as a function of x, y scatter, the DSC is displayed at a different scale so the scatter can be seen. b) Baseline 1 (distortion) error as a function of x, y scatter, c) Baseline 2 (spherical) error as a function of x, y scatter d) Baseline 3 (generic) error as a function of x, y scatter. It can be seen



a) DSC Error as a fcn of x, y Scatter

b) Baseline 1 (distortion) Error as a fcn. of x, y Scatter



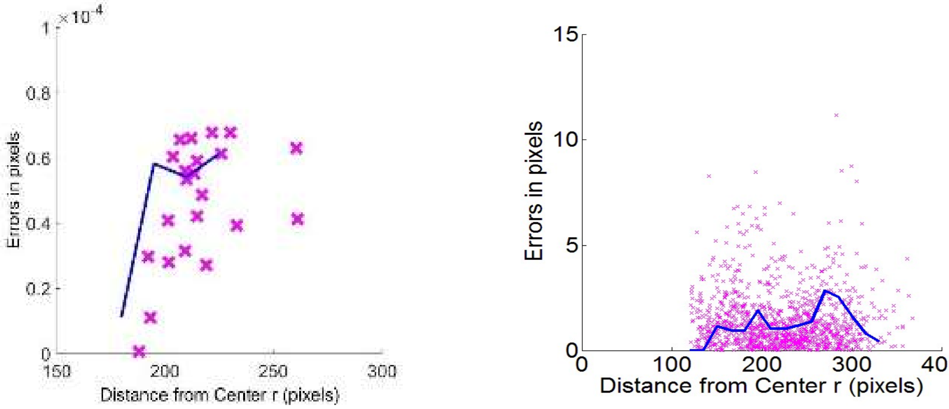
c) Baseline 2 (spherical) Error as a fcn. of x, y Scatter

d) Baseline 3 (generic) Error as a fcn of x, y Scatter

Figure 36. Error Scatter Plot

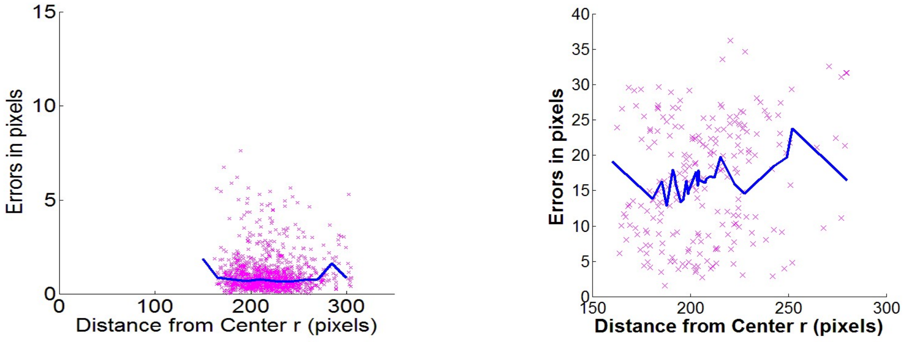
Comparison of the error as a function of x, y scatter for the four methods. Only the moderate data set is shown. a) DSC method (different scale to show scatter), b) Baseline 1 (distortion) Method, c) Baseline 2 (single viewpoint spherical) method, d) Baseline 3 (generic) method.

that the least error scatter is for the DSC method, the worst for Baseline 3 method then Baseline 1 and finally Baseline 2. Unlike the other methods, the DSC has one error point per calibration board, where the others have points for the total number of intersections in the calibration boards.



a) DSC Error as a function of Radius

b) Baseline 1 (distortion) Error as a function of Radius



c) Baseline 2 Error (single viewpoint spherical) as a function of Radius

d) Baseline 3 Error (generic) as a function of Radius

Figure 37. Error vs Radius Plot

Comparison of the error as a function of radius for the four methods. Only the moderate data set is shown. a) DSC method (different scale to show scatter), b) Baseline 1 (distortion) Method, c) Baseline 2 (spherical) method, d) Baseline 3 (generic) method The mean of the error scatter is overlaid as the blue curve.

Fig. 38 shows the error data as a function of radius for the four methodologies. The blue curve in Fig. 38 represents the median of the scatter data for each method. Fig. 38a) DSC error as a function of radius, the DSC is displayed at a different scale so the scatter can be seen. 38b) Baseline 1 (distortion) error as a function of radius, 38c) Baseline 2 (single viewpoint spherical) error as a function of radius 38d) Baseline 3 (generic) error as a function of radius. The blue curve in Fig. 38a) represents the median of the error data and is near zero for the DSC methodology. Again the DSC method has one point per calibration board. It can be seen that the DSC has a similar but smaller spread in angle, but an order of magnitude smaller spread in radius error.

The average error for Baseline 1 (distortion) is 2 pixels where the max scatter is up to 10 pixels, and Baseline 2 (single viewpoint spherical) average error is 0.94 pixels and the max error is about 7 pixels. The DSC error is sub pixel and on the order of a thousandth of a pixel. The DSC, Baseline 1 (distortion), and Baseline 2 (single viewpoint spherical) methods track fairly close in finding the center.

Baseline 1 and 2 methods found the center within 4 pixels, and the DSC was within 1 pixel. With the Baseline 3 (generic) method, the center is off by 68.8 pixels in the y direction. The actual center is about 324, 241 which is matched closest by the DSC method at 323.75, 240.86. There is a 5% delta in focal length. The uncertainty associated with the average errors is lowest for DSC at 0.00192 followed by BL2 at 0.05819, BL1 at 0.1011, and finally BL3 at 0.4707.

The DSC methodology is better than the other three Baseline methods with smaller error and less uncertainty. as shown in Table 15.

We have included in Table 17 a fourth column showing the Baseline 3 methodology [35,36] that had much higher errors compared to the other two Baseline methods and it didn’t work well with the IR data. Also, it can be seen in Fig. 37 that the Generic method’s error is on the order of 30 pixels and is much higher than the other two Baseline methods chosen for comparison. Also, Baseline 3 (Generic) missed the center by 68.8 pixels. It can be seen from Table 15. that the DSC method is the best performer.

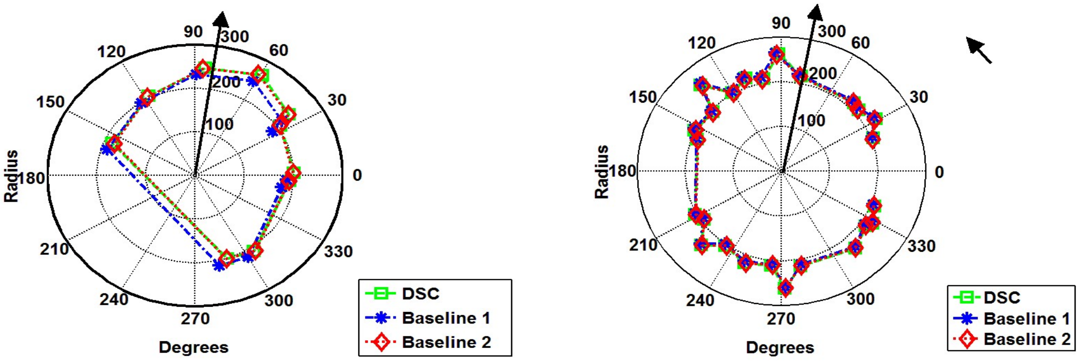
Table 17. Methodology Numerical Comparison

Framework	DSC	BL1 distortion	BL 2 single viewpoint spherical	BL3 generic
Average Error	0.00265	2.0057	0.94	16.5660
Standard Deviation	0.013	0.6928	0.3557	13.9032
Uncertainty	0.00192	0.1011	0.05819	0.4707
Calibration Time	0.017	2.09	53.43	325.2127
Center	323.75, 240.86	321.02, 239.96	323.50, 244.34	322.7239 309.8056
Focal Length	264.99, 264.99	N/A, N/A	251.01, 253.75	313.0189 253.9237

4.1.5. Optimal number selection of image capture of calibration boards

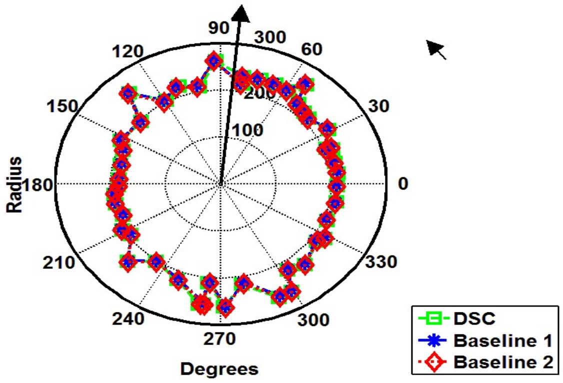
We did an analysis to determine the optimum number of calibration boards. We compared Ratio of Calibrated to Actual, error, and computation time. Fig. 39 is our plot of the three methodologies in polar form showing the relative comparison of the r, theta polar points obtained from the three methodologies. φ was nominally 90 degrees and relatively constant for all methods so it was not presented.

Fig 39 shows the $\boldsymbol{R}_{CA}$ in a comparison polar line plot for the moderate data set. The moderate case has the best match to the actual data for the four methods. The three Baseline methods did not originally present this data, but we added it to all of their routines for comparison. The polar plot shows the comparison of the best three methodologies in finding the r, θ (theta) points for each of the data sets. The baseline 3 (generic) method had an average 19% error (peak 64%) in r, θ compared to ground truth measurements, so it wasn't included in the polar plot. The sparse data set with only ten (10) calibration boards had the least consistency between the three methods. For the DSC and Baseline 1 and 2, all three matched the ground truth *r, θ* within about 5% with DSC at 0.05147, B1 at 0.04755, and B2 at 0. 05083. Both the moderate and dense track relatively closely with each other and with the *r, θ* pairs matching fairly well to ground truth. The dense set is slightly better.



a) Sparse data set

b) Moderate data set



c) Dense data set

Figure 38. Comparison Polar Plot

Comparison polar plot of the r, θ for DSC compared to the two Baseline methodologies. For a) sparse data 10, b) moderate data 25, and c) dense data 47 calibration board images. The main three data sets r, θ compare reasonably well with the ground truth with the delta being roughly 5% for all three.

The results of the values for R_{CA}, εd, t are given below in Table 18.

Table 18 shows the average R_{CA} for all four methods and data sets. Baseline 2 had the most consistency between data sets. The differences in R_{CA} were not sufficient to influence the final $f\ (R_{CA}, \epsilon_d, t)$ result.

Table 18. Ratio Calibrated to Actual(Rca) Comparison

R_{CA}	sparse	moderate	dense
DSC	1.1278	0.9783	0.97086
Baseline 1 (distortion)	2.0285	0.98198	0.9855
Baseline 2 (single viewpoint spherical)	1.1142	0.976033	0.9762
Baseline 3 (generic)	2.7058	1.05831	Failed

$\vert R_{CA-1\vert\vert}$	sparse	moderate	dense
DSC	0.089742	0.05147	0.0541
Baseline 1 (distortion)	1,0285	0.047553	0.0532
Baseline 2 (single viewpoint spherical)	0.1465	0.05083	0.0535
Baseline 3 (generic)	1.7058	0.19132	Failed

The results for R_{CA} are shown in Fig. 40 and both R_{CA} and $|R_{CA} - 1|$ are summarized in Table 18, which shows that the R_{CA} has the most variance for the sparse and dense cases, and the moderate is closest to actual for all four methods.

Both the moderate and dense are due to outliers with the bulk of the data being comparable to the sparse case. The difference between the results for the moderate and dense data sets (25 and 47 calibration boards) is so small that it isn't significant. Also, the larger number of calibration boards allows for room to throw out the outliers. This result shows that 10 images are adequate from an error standpoint. The benefit of 25 or 47 images is that if there are any problem images they can be deactivated and not used in the calibration.

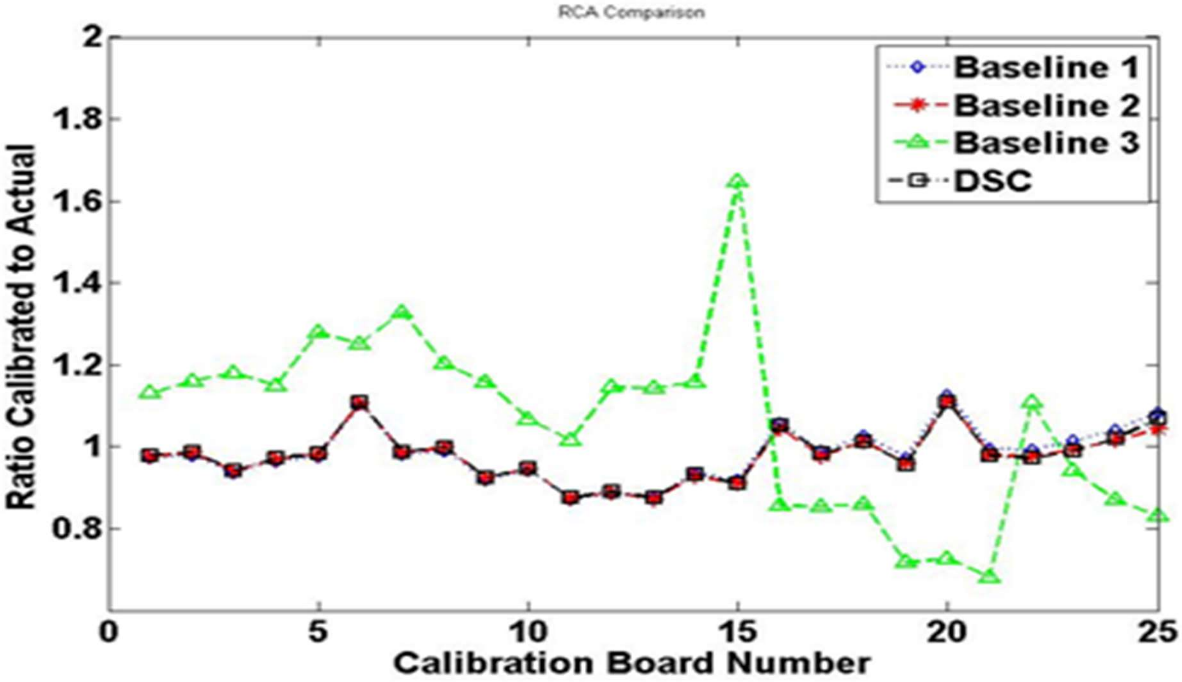


Figure 39. Ratio Calibrated to Actual (R_{CA})
Comparison ratio of calibrated to actual (R_{CA}) for DSC method compared to the three Baseline methodologies for the moderate data set.

The results for the three comparison methods error are summarized in Fig. 41. The best error result is the sparse data set at 10 images with 1.526 x 10-5 the dense at 0.00265, and the moderate at 0.0628-pixel error.

The maximum error of Baseline 1 (distortion) and 2 (single viewpoint spherical) is on the order of one to two pixels, the Baseline 3 is about 16 pixels, while the DSC method is sub-pixel and near zero. Of the three methods the DSC has the lowest error less than 0.01, followed by Baseline 2 (single viewpoint spherical) at about 1 pixel and Baseline 1 (distortion) at 1.5 to 2 pixels of error, and finally Baseline 3 (generic) at 16.566 pixels of error. The added complexity of Baseline 1 and Baseline 2 are not required as the simpler DSC method gets better accuracy and comparable calibration results.

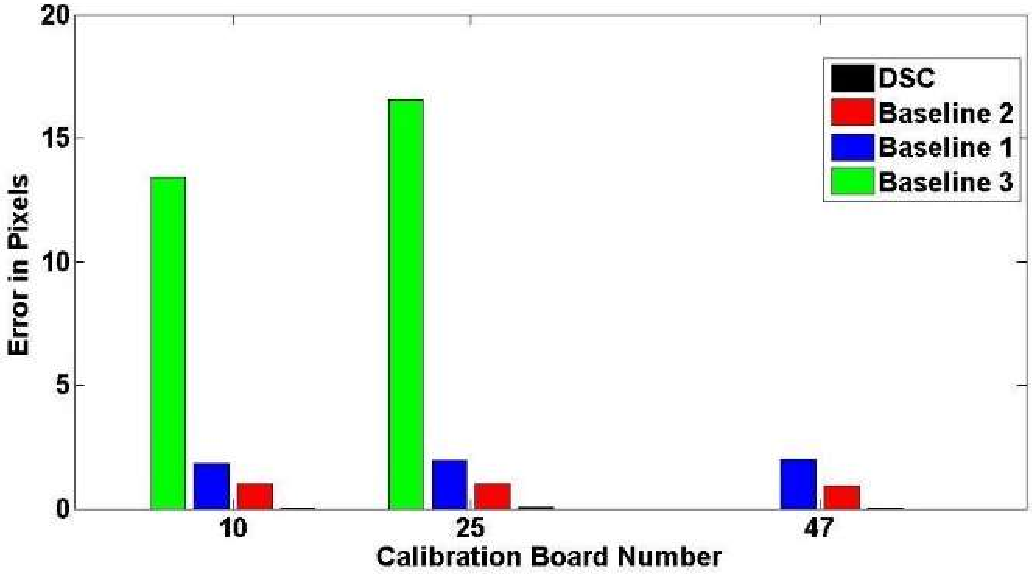


Figure 40. Methods - Error Bar Chart
A comparison plot of the error for the three data sets; sparse, moderate, and dense with 10, 25, and 47 images respectively. The plot compares the average error of the four methods with DSC being near zero, Baseline 1 (distortion) and 2 (single viewpoint spherical) on the order of one to two pixels. Baseline 3 (generic) between 14 and 16 pixels for sparse and moderate datasets However, the Baseline 3 method did not work for the Dense data set and did not converge.

Table 17 in Section 3.1.5 shows the comparison of error, standard deviation, computation time, center, and focal length for the four methodologies. Again DSC method has the lowest computation time at under 0.032 seconds for the sparse case, 0.084 for moderate case and 0.17 for the dense case. The DSC computation time is then followed by the Baseline 1 (distortion) methodology at about 2 seconds. The Baseline 2 (single viewpoint spherical) methodology had a computation time at 12 to 54 seconds. Finally, the Baseline 3 (generic) computation time was 933 seconds.

Optimizations utilizing the four methodologies for three values Ratio Calibrated to Actual $\boldsymbol{R_{CA}}$, error ϵ_d, and $\boldsymbol{t}$ are considered using (11) in Section 2.3.2. The results of Pareto optimization are shown in Fig. 42 and Table 19.

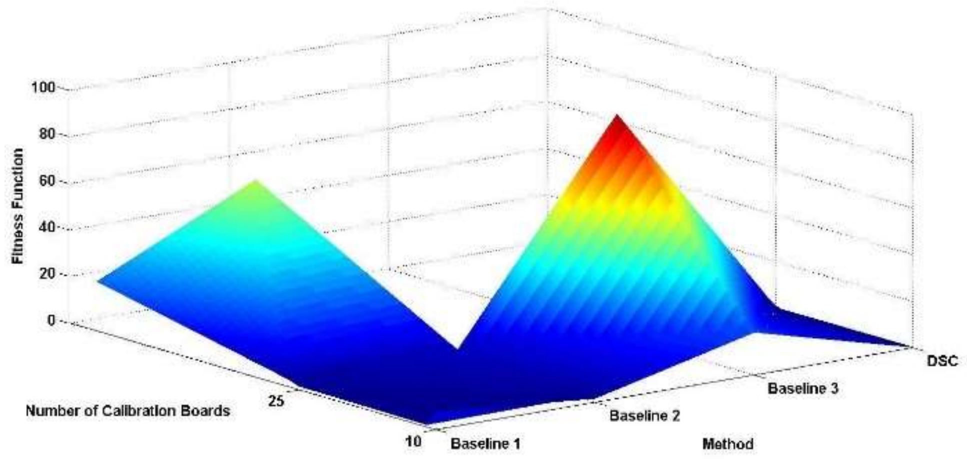


Figure 41. Pareto Optimization Plot
Plot of the Pareto Optimization results for the fitness function from Section III.C equation (12) for the four methodologies and three data sets.

Fig. 42 shows the plot of the Pareto Optimization for the fitness function (12) for the four methods and the three data sets. The plot results are summarized in Table 9 and it can be seen from the plot that the DSC method has the best results for the fitness function followed by Baseline 1, and 2 with Baseline 3 being the least fit. Table 19 results show that **$f (R_{CA}, \epsilon, t)$** was lowest for the DSC moderate data set. Under this image set, the Pareto-optimization provided a better RCA match, but the sparse case had the lowest error. This resulted in the overall lowest value for **$f (R_{CA}, \epsilon, t)$**. Thus the moderate data set is optimum for calibration in this case.

Table 19. Optimality Comparison

$f (R_{CA} , \epsilon, time)$	sparse	moderate	dense
DSC	0.11796	**0.0499**	0.05326
Baseline 1 (distortion)	4.0827	2.6580	21.4600
Baseline 2 (single viewpoint spherical)	1,969	6.4555	52.7600
Baseline 3 (generic)	18.130	94.70	Failed

4.2.Index Based and Thermal Region Fusion Experimental Results

The experimental results section follows the organization of section 4.2 and utilizes the 4.2.1. Describes the *Omni-direction Camera and Data Setting*; 4.2.2 Describes the results *Visual and IR Index Based Vegetation Detection*; 4.2.3 *IR Stream Segmentation using Region based Thermal Analysis* – Describes the results of the combination of the MNDVI and thermal region based stream segmentation approaches, using Region Based IR thresholding with thermal analysis and region growing threshold segmentation fusion of O-D IR and Visual Stream.

4.2.1. Omni-direction Camera and Data Setting

Data was captured from the O-D far-IR camera and Kinect camera systems; we later captured additional data from our O-D IR and O-D visual camera. The data sets are presented in Table 20. We then processed the data with the two approaches, the MNDVI, and the TRF fused, and compare their results. A representative sample was used from data set 1 (IR) and dataset 2 (Kinect) to produce Fig 34. O-D IR and O-D Visual samples from Data Set 3 and 4 were used to produce Fig 35. In Fig. 34 and 35, the visual image, the IR image and the Thermal Fused image are shown

Table 20. Camera Data Settings

Data Set	Image Type	Size	Sensor
1	far-IR O-D	640 x480	O-D IR
2	Visual Kinect	Various	Kinect Visual
3	O-D IR	640 x 480 Unwrapped to 181 x1760	O-D IR
4	O-D Visual	640 x 480 Unwrapped to 181 x1760	O-D Visual

along with the RGB and IR histograms. Both figures have the vegetation region threshold values superimposed. Fig. 43 and 44 second plot shows the spectral bins in the thermal signature. Fig. 43 using datasets 1 and 2, shows the relationship between the red, blue, and green bands and IR grayscale bins for a representative sample of a segment of the unwrapped O-D IR, and Kinect images. It also shows the fused image, and the histograms have the thresholds overlaid.

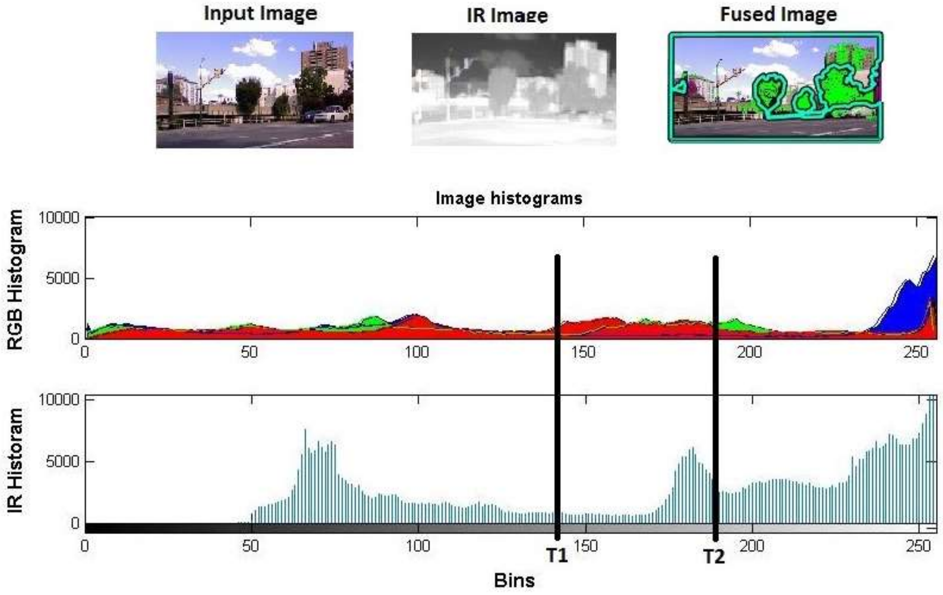


Figure 42. RGB and IR Histogram for O-D IR and Kinect Camera

Using datasets 1 and 2. A representative input Kinect visual and O-D IR images. The first plot is the histogram distribution of the blue, green and red bands of the input image shown above. The second plot

Fig. 44 using datasets 3 and 4 (o-D IR and O-D Visual) shows the same relationship for a representative unwrapped O-D color and O-D IR sensors along with the fused image showing the detected vegetation region. Again, Fig. 44 the RGB and Gray scale histograms with thresholds overlaid are shown.

4.2.2. Visual and IR Index Based Vegetation Detection

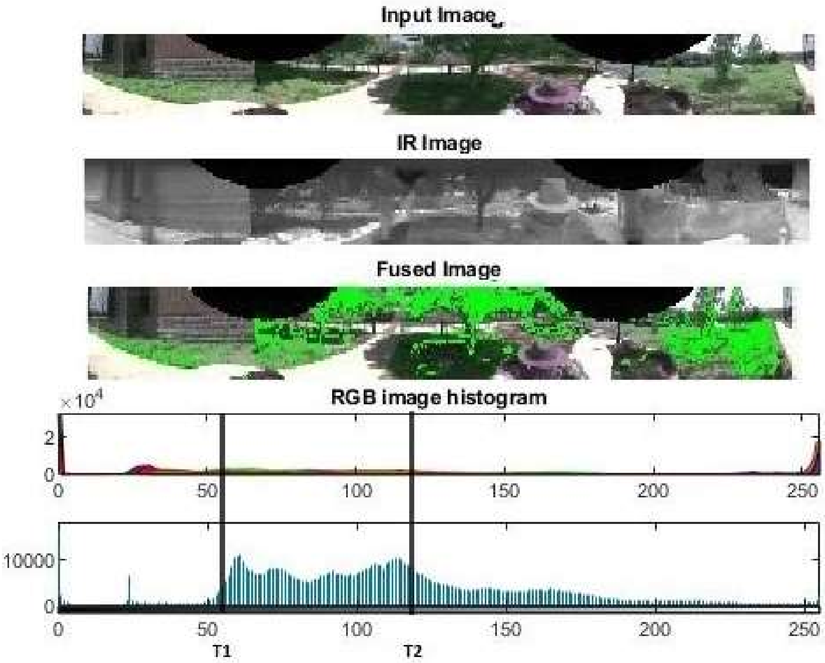


Figure 43. RGB and IR Histogram for O-D IR and O-D Visual Cameras

Using datasets 3 and 4. A representative unwrapped input O-D visual and O-D IR images. The first plot is the histogram distribution of the blue, green and red bands of the input image shown above. The second plot is the gray scale histogram of the IR image with the thresholds annotated.

In this section we present the results from the index based MNDVI approach. One of the key issues with the current implementations of vegetation index based processes is the false positives that it produces. It particularly struggles with synthetic materials that have high red absorption. The vegetation index approach has known failures in areas such as manmade materials and paints that have high red absorption and behave similar to vegetation. Since the index based approach compares the difference of the red band with the IR it can be seen how the MNDVI approach will be confused by this information. Applying MNDVI index based vegetation detection alone has a high incidence of false positives.

Fig. 45 and 46 compare MNDVI True Positives (TP) with TRF TP and MNDVI FP with TRF FP for the two different camera setups. We can see the relationship of true positives versus false positives in Fig. 46 for the O-D IR and Kinect cameras using datasets 1 and 2.

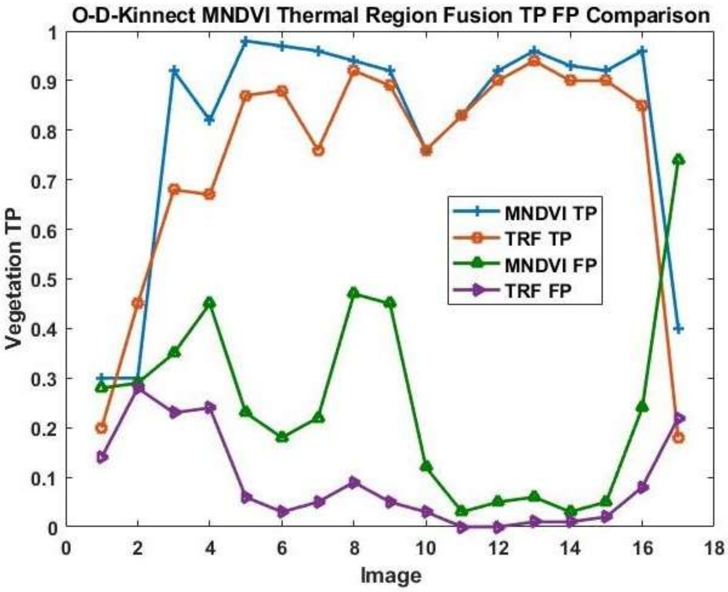


Figure 44. O-D IR and Kinnect MNDVI and TRF Comparison

O-D IR with Kinect cameras (Datasets 1 and 2) comparison of MNDVI and TRF true positive compared to false positive rates.

Similarly, we can see in Fig. 46 the relationship of true positives versus false positives for the O-D IR and O-D visual cameras.

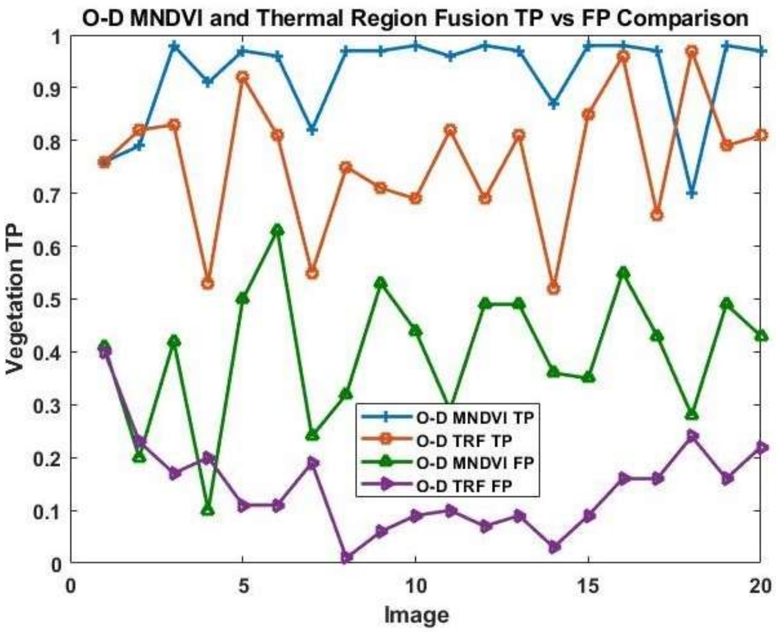


Figure 45. O-D IR and O-D Visual MNDVI and TRF Comparison

O-D IR and O-D visual cameras (Datasets 3 and 4) comparison of MNDVI and TRF true positive to false positive rates.

It can be seen in Table 21 that the MNDVI approach has an average accuracy of 86.74% of the true positives compared to the false positives at 32.5% with some as high as 40 – 70%. The visual MNDVI vegetation detection is shown in Row 1 - 3 of Table 21, and the fusion of the index based vegetation detection and the IR thermal based Region Fusion TRF results are shown in Row 4 - 6 of Table 21. As a result, we need a more robust method to detect the vegetation which we propose as our Deep Learning Fusion Network (DeepFuseNet) presented in Chapter 4.

4.2.3. IR Stream Segmentation using Region based Thermal Analysis

The IR Thermal Region Fusion results are shown in Fig. 45 and 46 above. Table 21. show the comparison of the MNDVI and TRF approach for the overall experiment averaging the O-D Kinect and O-D IR and O-D visual camera setup. Table 21. summarizes the results for the thermal region based analysis segmentation and fusion. The segmentation finds the regions of similar temperature which can then be fused with the index based results in section 3.4.2

The MNDVI approach has an average true positive rate of 86.75% average across all four data sets. However, the false positive rate is high with an average of 32.5% and a peak of 68.5%. The TRF approach has an average overall accuracy of 75.16% for the true positives compared to the average false positives at 11.5% with a peak as high as 40%. The TRF method has a better false positive rejection rate, but has less accurate recognition. The TRF true positive rate is mid-range of the NDVI spread in Table 19 and therefore this is not considered a major detractor. The authors believe that this degradation in performance is a result of the camera's not being co-linear with the same viewing axis. This could be improved with a better choice and design of the cameras. This could be further improved by the application of deep learning to the vegetation detection problem.

The ROC curves for the MNDVI and the TRF fused visual and far-IR streams to extract salient vegetation features is presented in Fig. 47 and 48.

Table 21. Thermal Segmented Region Fusion Results - Comparison of MNDVI to TRF

Attribute	Data Set	True % Positive	False % Positive	Ratio FP/TP	Peak FP/TP
MNDVI	1-2	81.12	25.00	0.3082	0.7400
MNDVI	3-4	92.35	39.72	0.4301	0.6300
MNDVI	**Avg.**	**86.74**	**32.5**	**0.3731**	**0.6850**
TRF	1-2	74.00	9.00	0.1216	0.2800
TRF	3-4	76.31	14.49	0.1899	0.3990
TRF	**Avg.**	**75.16**	**11.75**	**0.1563**	**0.3940**

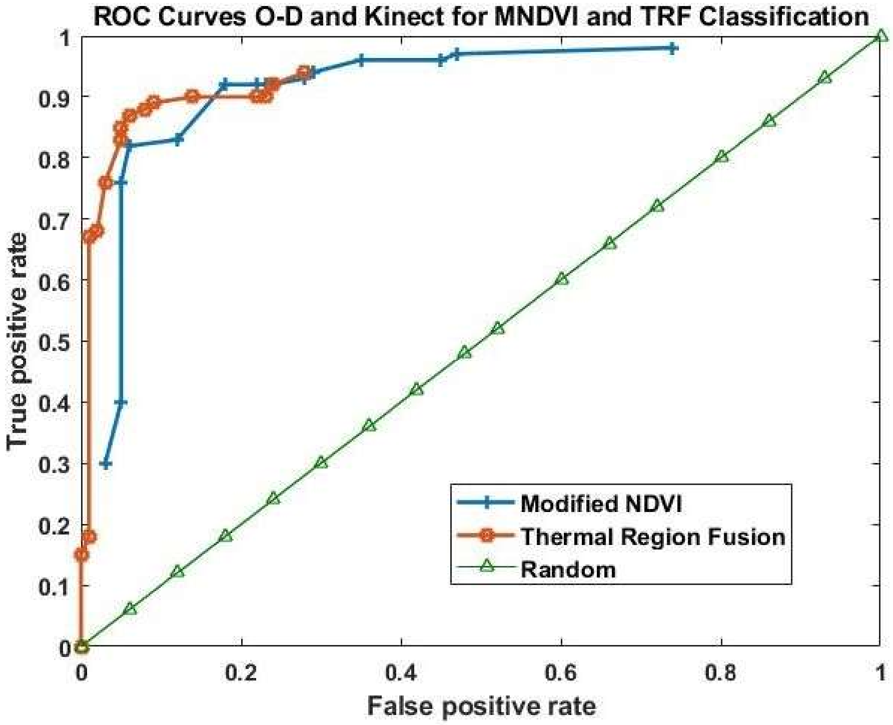


Figure 46. Receiver Operator Curve (ROC) O-D IR and Kinnect

ROC curves for the MNDVI and TRF approaches using the OD IR and Kinect cameras (Datasets 1 and 2). It can be seen that the TRF has fewer false detects but slightly worse recognition of the vegetation areas.

The TRF approach was better at false positives but had a lower true recognition rate, identifying the vegetation regions with fewer false positives than the baseline MNDVI approach. The positive

Table 22. Other Method Comparison Results

Comparison of Other Methods to our MNDVI to TRF				
Attribute	True % Positive	False % Positive	Ratio FP/TP	Peak FP/TP
Reflectance NDVI [51]	95.2	32.5	0.3412	0.8124
MODIS-NDVI [52]	72.6	35.5	0.4889	0,5510
NDVI[53]	47.7	61.29	1.2849	1.7442
MNDVI datasets 1 - 4	86.74	32.5	0.3747	0.8623
TRF *datasets 1 - 4*	75.16	11.5	0.1533	0.3940

impact of using TRF is that it effectively captures the vegetation pattern and fuses the two input streams from the O-D IR and vision cameras while rejecting false positives. The better overall performance is demonstrated by less false positives, and reasonable computation time. The Receiver Operating Curve (ROC) plots the true positive rate (TPR) or (sensitivity) against false positive rate (FPR) or (1-specificity). The ROC is commonly used to visualize the performance of a binary classifier. Fig. 47 shows the ROC curves for the MNDVI compared to the segmented TRF for the O-D IR and Kinect camera setup (Datasets 1 and 2), and Fig. 48 captures the ROC curves for the O-D IR and O-D visual camera setup (Datasets 3 and 4). It can be seen that the TRF thermal region segmentation also does not capture as much of the vegetation region as the MNDVI, but has fewer false detects. The ratio of false positive to true positive for MNDVI is 0.37 where TRF is 0.15. This again highlights the need for a new fusion method. It can be seen that the relationship between true positive and false positive is best for the TRF fusion approach.

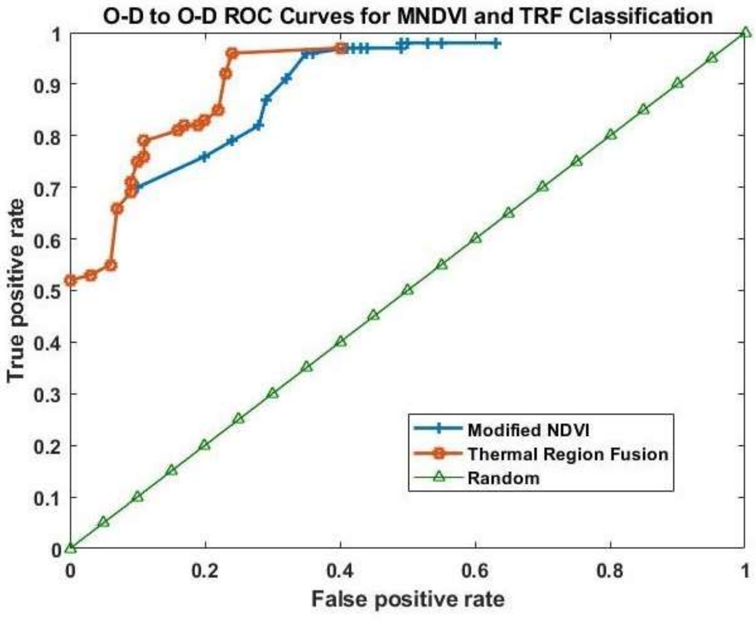


Figure 47. Receiver Operator Curve for O-D IR and O-D Visual
ROC curves for the MNDVI and TRF approaches using the O-D IR and O-D visual cameras (Datasets 3 and 4). It can be seen that the TRF has fewer false detects but slightly worse recognition of the vegetation areas.

Table 22 lists other methods and how we relate in this context. Our accuracies are consistent with the average of the other methods, however, our false positive rates are reduced. The first three references used provided no data from which we could estimate false positive rate, so the authors found three additional references with data. And have referenced their results in Table 22.

Fig. 38 summarizes the results in a bar chart showing the average percent across the datasets for True Positive (TP), percent False Positive (FP), and the ratio of FP/TP along with the peak FP/TP. Overall the TRF approach performs the best. However, the results still need improvement, leading us to the need for a deep learning approach applied in Chapter 4.3.

4.3.DeepFuseNet Experimental Results

The experimental results section follows the organization of section 4.2 and 4.3 and utilizes the 4.3.1. Data Setting, 4.3.2. Describes the results of using the baseline Far-infrared - Visual Modified Vegetation Index approach, 4.3.3. Describes the results of the combination of the MNDVI and thermal region based stream segmentation approaches, using Region Based IR thresholding with thermal analysis and region growing, and 4.3.4. Describes the results of stream fusion of O-D infrared and visual stream using our Autoencoder - CNN Fusion of O-D IR and Visual Stream.

4.3.1. Omni-direction Camera Setting

Table 23. Camera Data Settings

Data Set	Image Type	Size	Sensor
1	Far-infrared		O-D IR
2	Visual Kinect		Kinect
3	Visual O-D V		O-D V
4	ImageNet		Both
5	Web Scrapper		Both

Two sets of data were captured from the O-D Far-infrared camera and Kinect camera systems; we later captured additional visual data from our O-D visual camera. The data sets are presented in Table 23.

We then process the data with the tree approaches, the MNDVI, the Infrared Thermal based Region Segmentation (ITRS), and then Autoencoder-CNN fusion network, and compare the results.

4.3.2. Visual and Infrared MNDVI and TRF Vegetation Detection

In this section we present the results from the index based MNDVI and Thermal Region Fusion based approaches. The results are shown in Fig 49 and 50 and are summarized in Table 24. One of the key issues with the current implementations of vegetation index based processes is the false positives that it produces.

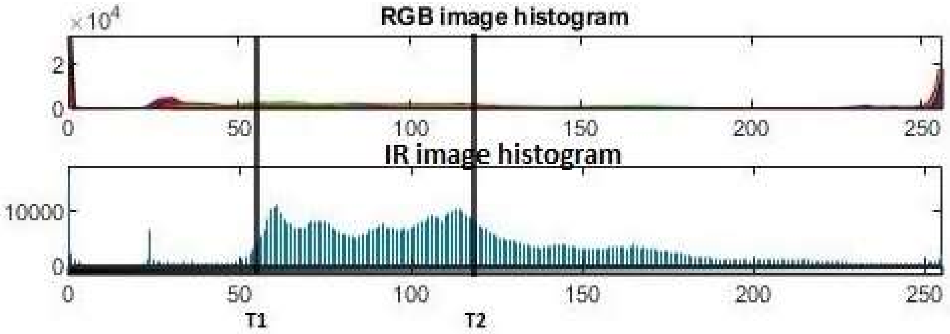


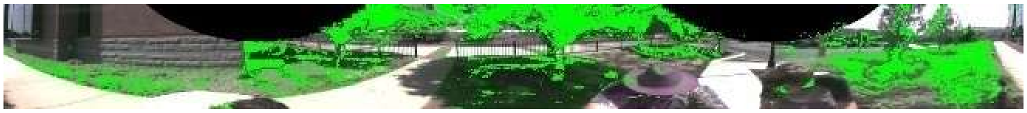
Figure 48. RGB and IR Histogram

Representative RGB and IR histogram for visual and IR images.

It particularly struggles with synthetic materials that are green in color. Fig. 50 the spectral relationship between the red, blue, and green bands for a representative sample. The vegetation index approach has known failures in areas such as manmade materials and paints that have high red absorption and behave similar to vegetation. Applying MNDVI index based vegetation detection alone has a high incidence of false positives.

a) Visual O-D image

b) IR O-D image



c) TRF fusion results

Figure 49. TRF Fusion Results

Fig. 50 is the results of the TRF method, visually showing the relationships of the results. It can be seen that while the TRF reduces the false Positives it does not completely capture the vegetation region. The ratio of false positive to true positive for MNDVI is 0.3749, and for TRF it is 0.1563. The loss in accuracy for the TRF method highlights the need for a new fusion method which is why we moved to deep learning approach. We can see the relationship of true positives versus false positives. The results are summarized in Table 24 for this section and the next two sections.

It can be seen that the MNDVI approach has an accuracy of 85.6% of the true positives compared to the false positives at 32.5. The TRF approach has an accuracy of 75.16% of the true positives compared to the false positives at 11.75%. %. From this and our work in chapter 3, we need a more robust method to detect the vegetation.

4.3.3. Autoencoder - Convolution Neural Network Fusion of O-D IR and Visual Stream

The application of the Autoencoder Feature Extractor applies an Autoencoder and Bottleneck CNN to extract salient features from the image and the training and loss results are presented in Fig. 51.

The unsupervised learning SAFE-CNN Autoencoder approach captured the features well, it was not as effective when trained with the fusion network. The results are shown in Fig. 51 and the training is only achieving an accuracy of about 80%. We are continuing to explore why this occurred and how we can fine-tune the model to better utilize these features.

The positive impact of using unsupervised learning with two Sparse Autoencoder Neural Network Feature

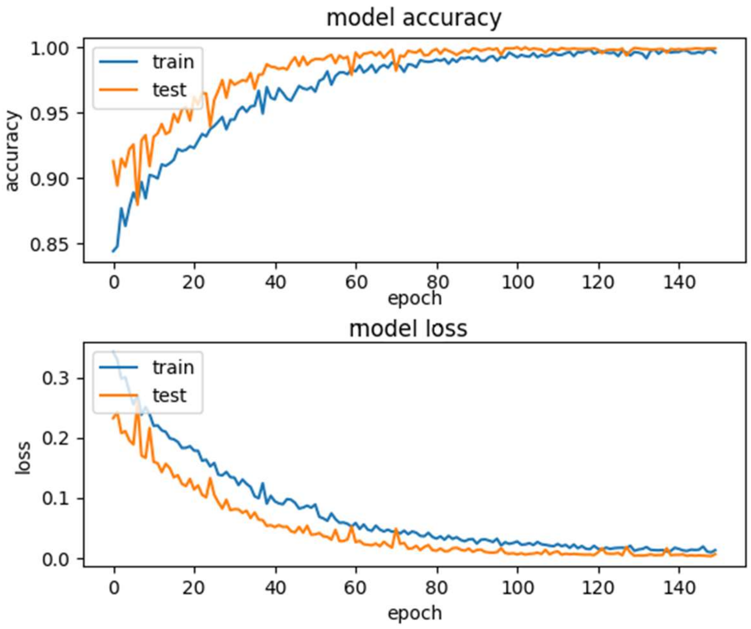


Figure 50. VGG16 Autoencoder Accuracy and Loss

Training loss and accuracy for VGG16 Autoencoder trained on ImageNet. Dataset[42] and fine-tuned on our vegetation set in order to extract vegetation features.

Extractors (SAFE) into a CNN is that it was hypothesized to more effectively captures the vegetation pattern. Once successful we believe this will provide better performance and less false positives.

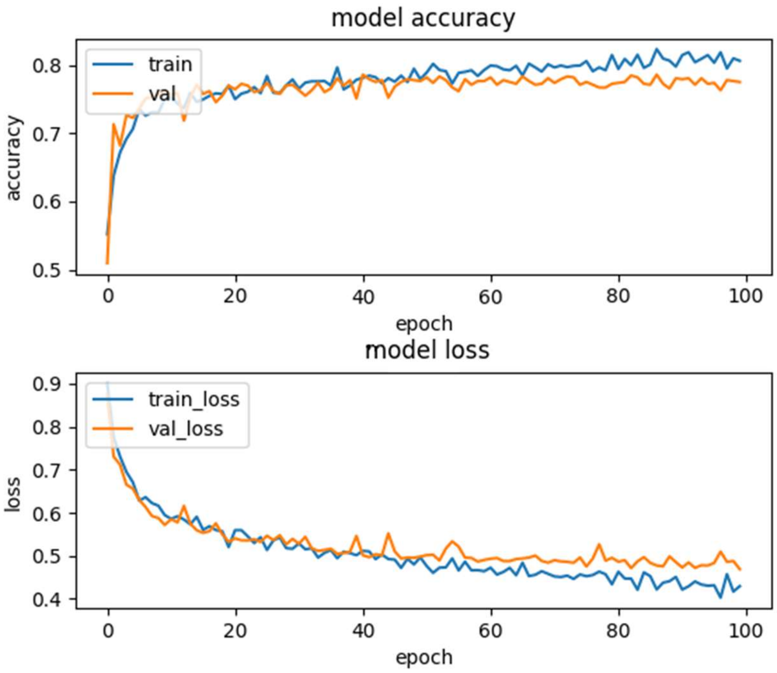


Fig. 15

Figure 51. Autoencoder Fusion Accuracy and Loss

Fusion model trained with Autoencoder input.

4.3.4. Convolution Neural Net (CNN)

Since we did not have a large dataset from which to train, we applied transfer learning to the problem. We utilized the large ImageNet [42] data set with over 1.2 million images and 1000 classes to get an initial trained model and then applied transfer learning and fine-tuning to refine the model to our smaller dataset. The transfer learning process freezes the weights of the lower layers of the network and adds a new top fully connected section which is trained on the smaller data set. This allows the network to retain the learned features of the lower activation filters, and refine the learning based on the new smaller data set. We looked at several models to determine the best for our transfer learning experiment.

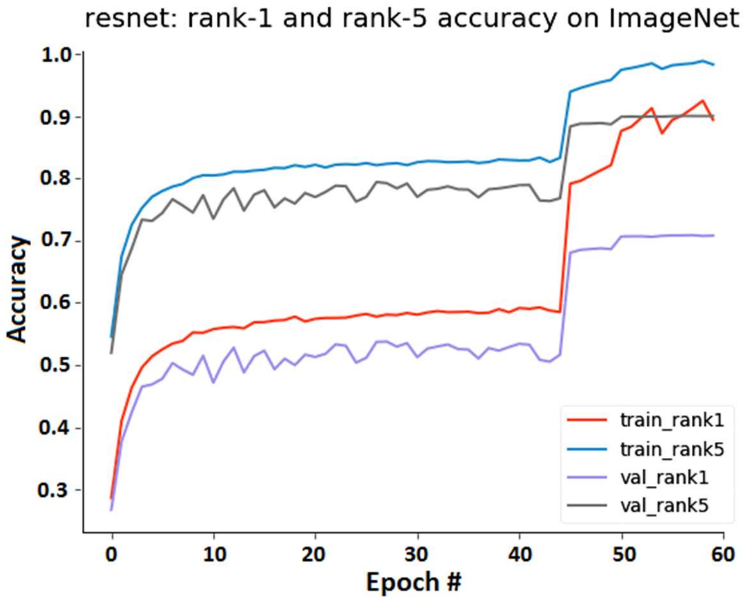


Figure 52. ResNet Accuracy and Loss Fine-tuning

Training loss and accuracy for ResNet trained on ImageNet data set.

Fig. 53 is a representative model trained on ImageNet showing the loss and accuracy after fine tuning the learning rate starting at 1e-3, 1e-4 and finally 1e-5. We initially used a bottleneck CNN as a feature extractor for the starting weights of our process. Fig. 54 shows results for the Bottleneck Feature Extractor (BFE). It can be seen that the BFE does not perform as well as the SAFE only capturing Fig. 54 shows the training / test accuracy and loss for the VGG16 based bottleneck feature extractor. The BFE only achieves 94% feature extraction compared to

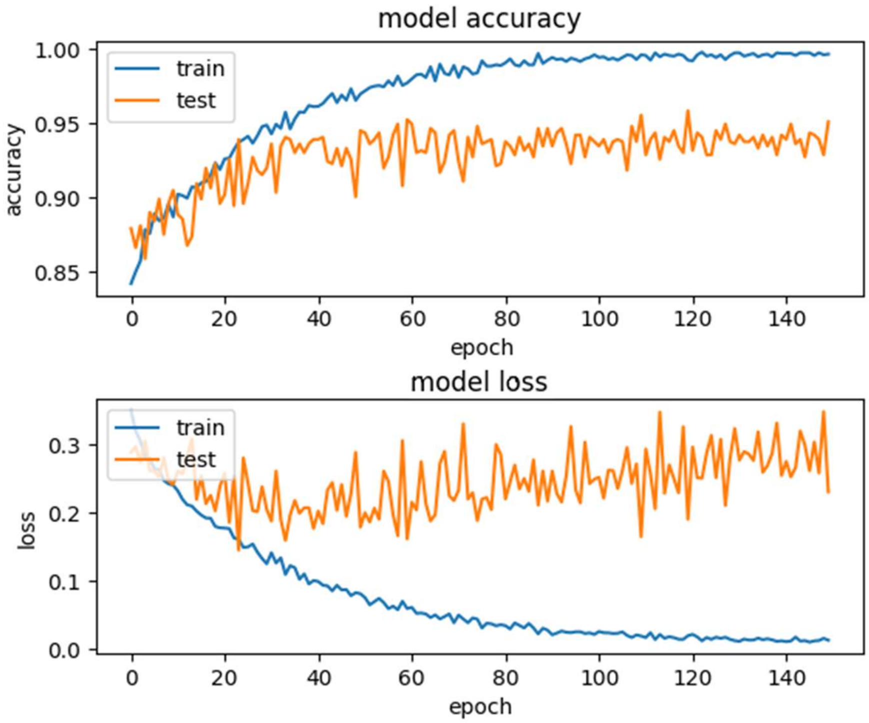


Figure 53. VGG16 Bottleneck Feature Extractor
Training loss and accuracy for VGG16 based bottleneck feature extractor leveraging ImageNet transfer learning and fine tune trained on our vegetation dataset.

the 99% for the SAFE approach.

The comparison results for the AlexNet, SqueezeNet, ResNet50, ResNet101, VGG16, and ResNet 152 models evaluated are presented in Fig 55, 56, and 57. We trained several models to find the best performance for our transfer learning experiment. Fig. 55 is the Rank 1 accuracy. The Rank 1 accuracy is the percentage of the assessments that the target is in the highest probability class prediction. Based on our literature search we initially were thinking that we wanted to use the deeper networks of ResNet50 or ResNet101. However, after this comparison we saw that ResNet50, ResNet101 and VGG16 all had similar performance with VGG16 having less overfitting.

AlexNet and SqueezeNet had lower performance than the other models. We then looked at the deeper ResNet models, and varied the momentum to see if that would time. The Rank 1 results for the four models we were evaluating were in the range of 70-72% accuracy. For these assessments we held the learning rate the same (1e-3) and initially compared all the models at a Momentum = 0.9. There were slight increases but not significant enough to accept the increased processing.

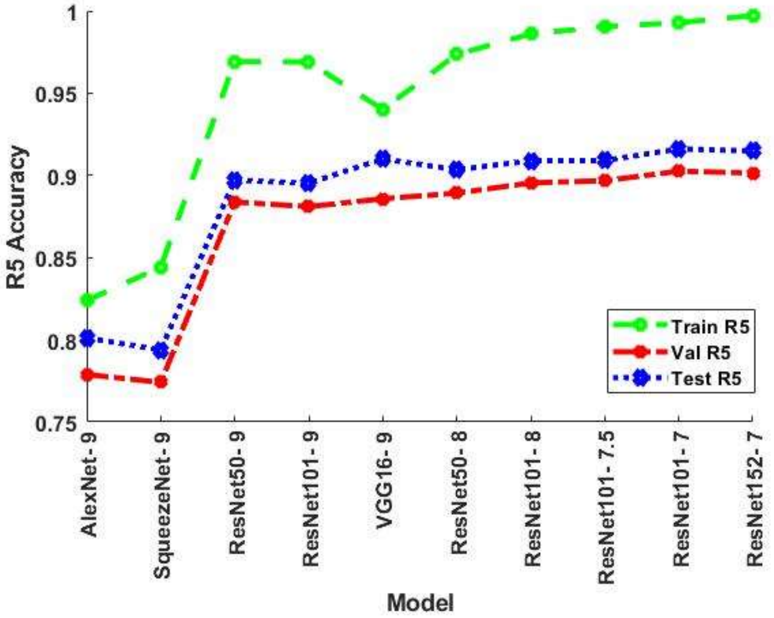


Figure 54. R5 Accuracy Comparison

R5 Accuracy across the various models with different momentum.

Fig. 56 is the Rank 5 accuracy. The Rank 5 accuracy is the percentage of the assessments that the target is in the top five highest probability predictions. Again the results were fairly repeatable with the accuracy ranging from about 90-92%.

Fig. 57 presents the training and validation Cross-Entropy Loss for the models being evaluated. Only the training and validation loss are shown since test is a prediction based on input image and does not have a loss component. Again other than AlexNet and SqueezeNet, the performance is very comparable. Again there were slight improvements for the deeper models at Momentum = 0.75 and 0.70. The performance for the deeper models is consistent.

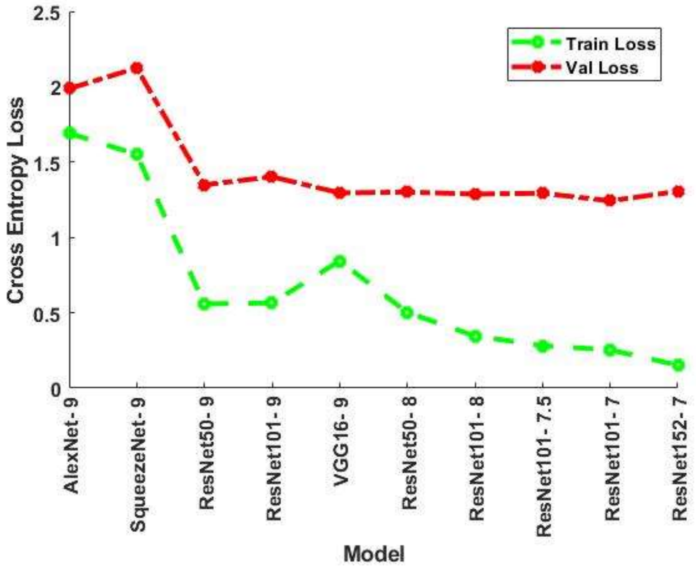


Figure 55. Cross-Entropy Loss Comparison

Cross-Entropy Loss across the various models with different momentum.

We did a number of fine-tuning experiments with VGG16, ResNet50 and ResNet101 and settled in on using VGG16 for our transfer learning and fine-tuning adaptation. We initially trained the models on ImageNet data and then did transfer learning to train on our smaller data set. The results will be presented in the next section.

4.3.5. DeepFuseNet

For our DeepFuseNet approach, we merged two VGG16 models one for the visual images and one for the IR images. The outputs of the two models were then concatenated and provided into two fully connect Dense layers and a softmax classifier.

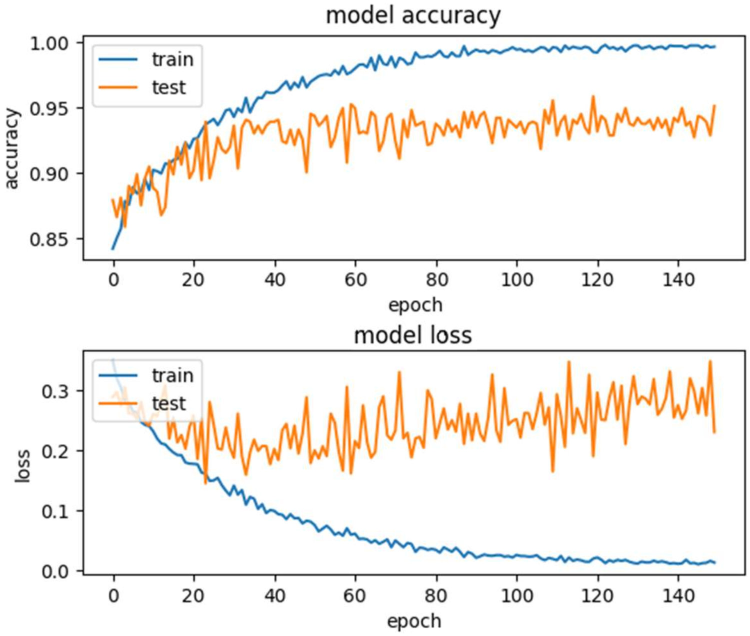
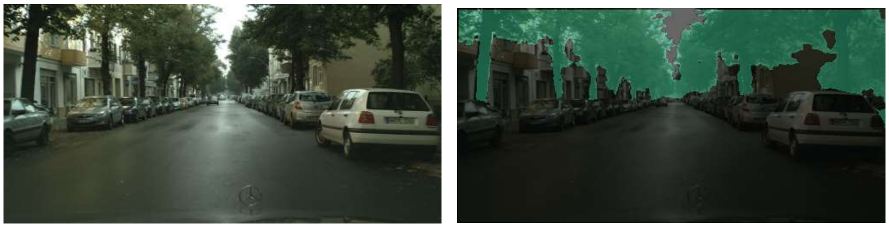


Figure 56. Fine-tuned Merged Model

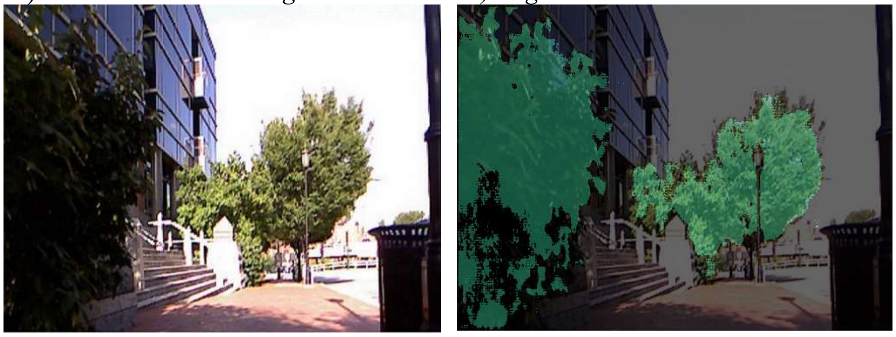
Fine-tuned merged model trained on vegetation data.

Fig. 58 shows the training results for the merged model. The training accuracy achieves 95% in 50 epochs. The validation test accuracy is about 92%.

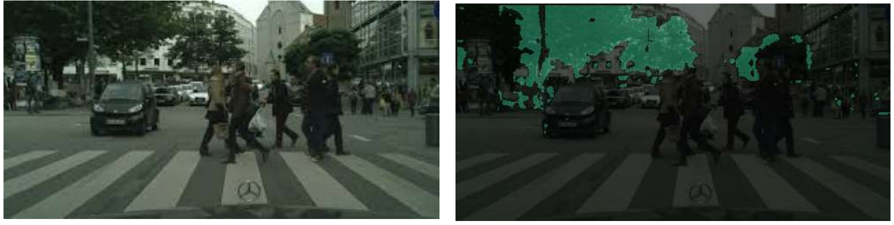
Fig. 59 shows representative samples of vegetation recognition. It can be seen that the capture accuracy is high at 95.6%. The false positives are very much improved over the baseline MNDVI approach or the TRF approach reducing it to on the order of 1-2%.



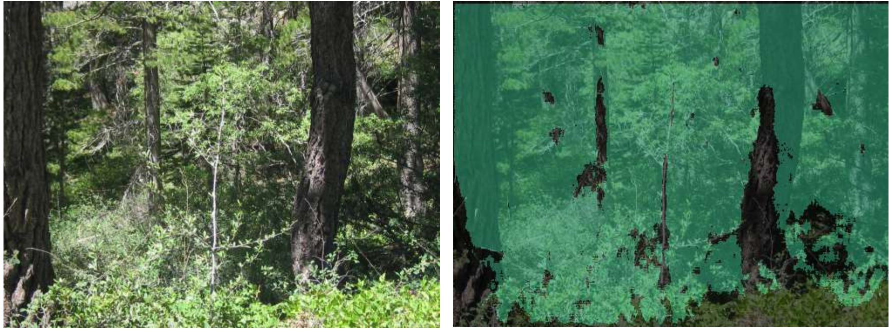
a) *Tree lined street image* *b) Vegetation Mask 1*



c) *VCU entrance image 2* *d) Vegetation Mask 2*



d) City scape image3 *e) Vegetation Mask 3*



e) Forest image 4 *g) Vegetation Mask 4*

Figure 57. Vegetation Detection Results
Vegetation detection results.

Table 24. Baseline and DeepFuseNet Results

	ACC	Loss	Val ACC	Val Loss	% False Positive
MNDVI	86.74	N/A	N/A	N/A	32.5
TRF	75.16	N/A	N/A	N/A	11.5
DeepFuseNet	95.6	0.1	92	0.2	1-2

Table 24 shows the comparison between the base MNDVI approaches, the union fused MNDVI and IR Vegetation Region (ITVR), the SAFE-CNN, and the DeepFuseNet approaches. The visual MNDVI vegetation detection is shown in row 1 of Table 24, and the union fusion of the index based vegetation detection and the Infrared thermal based Region Segmentation (ITRS) results are shown in row 2 of Table 24, and finally the SAFE-CNN and DeepFuseNet results are shown in the last two rows of the table. The relationship between true positive and false positive is found to be best for the DeepFuseNet fusion approach. The DeepFuseNet has an improvement in accuracy finding 95.6% of the true positives and reducing the false positives to less than 2%, which is an 16x improvement over the index based alone.

5. Chapter – Conclusion

The conclusion chapter includes four sections; one for each of the evaluated methods as follows. In Section 5.1, the conclusion for the Direct Spherical Calibration method is presented. In Section 5.2 is the conclusion for the index based and region segmentation based methods is presented. In Section 5.3 the conclusion for the Deep Fusion Network (DeepFuseNet) is presented. Finally, in Section 5.4 The summary conclusion is presented.

5.1. Direct Spherical Calibration (DSC) Conclusion

We developed a Direct Spherical Calibration (DSC) methodology which used the four corners of the calibration board to find the center of the calibration board, and then used the direct spherical coordinates of the center of the calibration board as the re-projection points matrix was used to iteratively find the calibration parameters. Of the four methods evaluated, the DSC method had the least time, a simpler corner extraction methodology, more reliable calibration board coordinate capture, and sub pixel accuracy compared to the other three Baseline methodologies. Due to the robustness and the simplification of the DSC, it was demonstrated that it is not necessary to use the more complex methods.

When using the DSC methodology, it was found that the lowest error could be found with the sparse data set. Both the moderate and dense data sets had outliers caused by the occlusion of some of the grid points and poor IR edge quality. The advantage of the larger data sets would be the ability to throw out outliers and possibly achieve better performance as shown in the Pareto optimization analysis. However, the DSC with moderate data set had the overall optimum performance.

The elimination of outliers was not explored in this research. The results from this calibration implementation will be applied in our follow on research utilizing the omnidirectional IR and visual cameras. This calibration approach is required as a foundation for our further research into the application of these instruments for the measurement of scene material visual properties through the fusion of the O-D IR and visual streams, and ultimately to the use of a convolution neural network deep learning approach to a semantic scene reconstruction model.

5.2.Index Based and Thermal Region Fusion Methods Conclusion

We presented a TRF Fusion approach for O-D IR and vision stream and compared the results to our modified normalized difference vegetation index (MNDVI) approach and a fusion of the MNDVI and IR Thermal Region based approaches to detect and classify vegetation. Table 20 summarizes the baseline sensor fusion results with MNDVI having an 86.73% detect rate, but it had a 32.5 % False Positive rate. The IR Thermal Region (IRTR) fused with the visual index based lowered the false detect rate to 11.5%, but was not as good at detecting vegetation dropping to 75.16%. The method of determining the thermal region was a threshold region growing and segmentation. We demonstrated a 64 % improvement in false positive but resulted in a 14.5% reduction in true positive with our TRF fusion approach. The reduction in true positive results is still mid-range of the NDVI results presented in [49] and Table 22. The authors hypothesize that the reduction in true positive recognition was due to the two cameras not having the same vertical viewing axis. Also, the IR and Visual cameras did not have the same vertical viewing angle and could not be fully registered. While our TRF approach made a significant reduction in false positives, it lost ground in true positive recognition. However, the authors feel that these proof of concept results merit further work in this area. The approach had reasonable performance in varying lighting conditions, but was not robust to the different viewing angle of the two cameras. This demonstrates a need to use a learning approach that will address the issue of camera quality.

The follow on work will fuse the O-D IR and O-D Visual streams utilizing the application of deep learning through Convolution Neural Networks and context based reasoning algorithms to discriminate between objects in a spatial scene (grass-wall, rock-bush, puddle-hole, and door-window). This will then be applied to a robot platform for object detection classification and tracking.

5.3.DeepFuseNet Conclusion

The three methods ordered from worst to best are MNDVI, TRF, and DeepFuseNet. It is clear that DeepFuseNet provides the best average results The Sparse Autoencoder Feature Extractor and CNN Fusion approach did not give good initial results so we are still exploring that approach and will present it in a future paper. results to our modified normalized difference vegetation index (MNDVI) approach and a union based fusion of the MNDVI and Infrared Thermal Region based approaches to detect and classify vegetation. We demonstrated a 95.6% accuracy for true positives and a 93.8% reduction in false positive with our DeepFuseNet approach. We have adapted the approach to our pioneer robot platform for follow on experiments.

The application of the BFE extractor and the Deep CNN fusion of the O-D infrared and visual streams have laid a foundation for a significant improvement in the intelligent perception of robot platforms. This is due to deep layers of feature encoding that employ a large spatial context for labelling the vegetation and non-vegetation.

Future studies will refine the method, further explore the SAFE-CNN approach and also use texture analysis and context based reasoning algorithms to discriminate between objects in a spatial scene (grass-wall, rock-bush, puddle-hole, door-window). This will then be applied to a robot platform for object detection classification and tracking.

5.4. Summary Conclusion

In this dissertation, we propose an intelligent visual perception method for an unmanned ground vehicle (UGV) utilizing the Deep Sensor Fusion of an O-D IR sensor and an O-D visual sensor. The O-D IR and O-D visual sensors provide for a wide field of view (FOV) for the UGV to detect objects in the entire 360-degree environment around the platform. The O-D IR sensor allows for the detection of thermal signature features in the image and the O-D visual sensor allows for the detection of color and texture features in the image.

Table 25. Summary of Methodologies

Methodology	Contribution	Data Type
Direct Spherical Calibration (DSC)	Improved calibration capture Sub-pixel accuracy	OD-IR calibration grids in three data sets,
Modified Normalized Difference Vegetation Index MNDVI	86.73% detect rate, with a 32.5% False Positive rate.	O-D IR – Kinect O-D IR – O-D visual
Thermal Region Fusion (TRF)	75.16% detect rate, with a with a 11/5% False Positive rate.	O-D IR – Kinect O-D IR – O-D visual
Deep Fusion Network (DeepFuseNet)	95.6% detect rate, with a with a 2% False Positive rate.	O-D IR – Kinect O-D IR – O-D visual

The features and advantages of each methodology are listed below:

Direct Spherical Calibration (DSC)

- O-D IR sensor provides thermal signature information from the scene and a wide FOV.
- O-D IR Sensor has more noise and less resolution than the standard visual sensors used by UGV
- Developed a methodology which used the four corners of the calibration board to find the center of the calibration board, and then used the direct spherical coordinates of the center of the calibration board.
- The re-projection points matrix was used to iteratively find the calibration parameters.
- Compared to three baseline methods, the DSC method had the least time, a simpler corner extraction methodology, more reliable calibration board coordinate capture.
- DSC method resulted in sub pixel accuracy compared to the other three Baseline methodologies at 2 or greater pixel accuracy.

- Optimality Analysis showed that the optimum calibration performance was achieved by the DSC methodology and the moderate (25) dataset.

Modified Normalized Difference Vegetation Index (MNDVI)

- Normalized Difference Vegetation Index approach to vegetation detection has high false positive rates.
- Modified NDVI from near IR to far IR
- MNDVI results in an 86.73% detect rate, but it had a 32.5 % False Positive rate.

Thermal Region Fusion (TRF)

- We presented a Thermal Region Fusion (TRF) approach for O-D IR and vision stream and compared the results to our modified normalized difference vegetation index (MNDVI).
- Fusion of the MNDVI and IR Thermal Region based approaches to detect and classify vegetation.
- The method of determining the thermal region was a threshold region growing and segmentation.
- TRF results in an 75.16% detect rate, but it had a 11.5 % False Positive rate.
- Demonstrated a 64 % improvement in false positive but resulted in a 14.5% reduction in true positive.
- The reduction in true positive results is still mid-range of the NDVI results presented in [49] and Table IX.
- The authors hypothesize that the reduction in true positive recognition was due to the two cameras not having the same vertical viewing axis. Also, the IR and Visual cameras did not have the same vertical viewing angle and could not be fully registered.
- This demonstrates a need to use a learning approach that will address the issue of camera

quality.

Deep Fusion Network (DeepFuseNet)

- Fused the O-D IR and O-D Visual streams utilizing the application of deep learning through Convolution Neural Networks.
- The combination of the O-D IR and O-D visual sensors allowed for a richer feature vector to be extracted from the data.
- In our work we looked at vegetation index based and region fusion methods and then compared them to our DeepFuseNet approach.
- DeepFuseNet approach achieved a 95.6% true positive recognition rate while also showing a 92% reduction in false positives over the classical NDVI approach.
- Our first step was to look at feature extraction to establish the feature vector to be fed into the network.
- We looked at two approaches for the feature extractor. First an Autoencoder feature extractor, and second a VGG16 Bottleneck feature extractor.
- We found that the Autoencoder Feature Extractor didn't perform as well in the overall system as the Bottleneck Feature Extractor.
- We then followed the feature extractor by a residual network utilizing transfer learning from the ImageNet data set to our smaller data set.
- The next step was the fine-tuning of the network.
- The final was a fully connected network to predict the region areas and concatenate the final feature mask.

The project described in this book has resulted in the following three major contributions:

1. Development of an improved Direct Spherical Calibration for low resolution O-D IR cameras..

2. Comparison of MNDVI, TRF and DeepFuseNet results.

3. Accomplishment of a high level of vegetation detection and false positive rejection by using more efficient transfer learning, fine-tuning, and deep learning when compared to classical index based vegetation detection.

For our future work, we plan to implement additional deep learning approaches such as residual, recurrent, and semantic segmentation models to evaluate their application to enhanced robotic vision. We will also extend and evaluate the efficacy of the approach while the robot is moving in its environment.

APPENDIX A

REPRESENTATIVE MATLAB AND PYTHON CODES

The source codes are implemented in the platform of MATLAB® 7.10.0, R2017a(*) and Python 3.6. The MATLAB codes developed are built on top of the Image Processing Toolbox Version 2.10(*). *http://www.mathworks.com.

The M-codes in Appendix A include some comments, and extract the core parts of M-codes, representing the corresponding algorithms in the dissertation. The Python codes in Appendix A are built on top of KERAS and Tensorflow-gpu as a backend.

The Chapter 2 source codes for Direct Spherical Calibration are as follows:

A.1 Calibration GUI (omni_calib_gui_normal_s.m)
A.2 Go_omni_calib (go_omni_calib_itr_s.m)

A.3 Spherical Coordinates (Spherical_s.m)
A.4 Multi objective optimization (simple_multiobj2.m)
A.5 Compute Omnidirectional Error (comp_omni_error_s.m)
A.6 Error Analysis (analyse_err_s.m)

A.1 Calibration GUI (omni_calib_gui_normal_s.m)

```
function omni_calib_gui_normal_s
missing = 1;
cell_list = {};
%-------- Begin editable region -------------%
%-------- Begin editable region -------------%
%
%       Modified by David Stone Apr 2014
fig_number = 1;
title_figure = 'Omni Camera Calibration Toolbox - Improved Version';
cell_list{1,1} = {'Mirror type','paramEst = mirror_type();'};
cell_list{1,2} = {'Load images','images = data_calib();'};
cell_list{1,3} = {'Estimate camera intri.',['check_border_estimate;'... ...
                  'if ~missing '...
                  '[gen_KK_est,borderInfo] =' ...
                  ' border_estimate(images,paramEst);'...
                  'end' ]};
cell_list{1,4} = {'Extract grid corners',['tic;','check_click_calib;'...
                  'if ~missing '...
                  '[gridInfo,paramEst] ='...
                  'click_calib_s(images,gen_KK_est,gridInfo,paramEst);'...
                  'toc;'...
        'end']};

cell_list{2,1} = {'Draw Grid Estimate',['check_click_calib;'...
                 'if ~missing '...
                 'drawImageWithPoints_s(images,gen_KK_est,gridInfo,paramEst);'...
                 'end']};
cell_list{2,2} = {'Calibration',['check_calib_optim;biased_calib=1;' ...
                  'if ~missing '...
        'tic;'...
                  'paramEst =
go_omni_calib_optim_iter_s(minInfo,images,gen_KK_est,gridInfo,paramEst);'...
                  'toc;'...
        'end']};
cell_list{2,3} = {'Rm Calibration',['check_calib_optim;'...
                  'if ~missing '...
                  'paramEst = rm_calib(paramEst);'...
                  'end']};
cell_list{2,4} = {'Analyse error',['check_calib_optim;'...
```

```
                    'if ~missing '...
                    'analyse_error_s(images,gridInfo,paramEst);'...
                    'end']};

cell_list{3,1} = {'Recomp. corners',['check_calib_optim;'...
                    'if ~missing '...
                    '[gridInfo,paramEst] ='...
                    'recomp_corner_calib_s(images,gen_KK_est,gridInfo,paramEst);'...
                    'end']};
cell_list{3,2} = {'Draw 3D',['check_calib_optim;'...
                    'if ~missing '...
                    'drawGrids3D_s(images,gen_KK_est,gridInfo,paramEst);'...
                    'end']};
cell_list{3,3} = {'Plot Pix/Rad errors',['check_calib_optim;'...
                    'if ~missing '...
                    'plot_omni_error_rho_s(images,gen_KK_est,gridInfo,paramEst);'...
                    'end']};
cell_list{3,4} = {'Add/Suppress images','images = add_suppress(images);'};

cell_list{4,1} = {'Show calib
results','show_calib_results_s(images,gen_KK_est,paramEst,gridInfo);'};
cell_list{4,2} =
{'Save','save_omni_calib(minInfo,borderInfo,images,gen_KK_est,gridInfo,paramEst);'};
cell_list{4,3} = {'Load','load_omni_calib;'};
cell_list{4,4} = {'Spherical', 'spherical_s;'};
cell_list{5,1} = {'Exit',['disp("Bye. To run again, type omni_calib_gui_s."); close('
num2str(fig_number) ');']}; %{'Exit','calib_gui;'};

show_window(cell_list,fig_number,title_figure);
%-------- End editable region -------------%
%------- DO NOT EDIT ANYTHING BELOW THIS LINE -----------%

function
show_window(cell_list,fig_number,title_figure,x_size,y_size,gap_x,font_name,font_size)

if ~exist('cell_list'),
   error('No description of the functions');
end;

if ~exist('fig_number'),
   fig_number = 1;
end;
if ~exist('title_figure'),
   title_figure = ";
end;
if ~exist('x_size'),
    x_size = 125;
end;
if ~exist('y_size'),
    y_size = 16;
end;
if ~exist('gap_x'),
    gap_x = 0;
end;
if ~exist('font_name'),
    font_name = 'clean';
end;
if ~exist('font_size'),
    font_size = 8;
end;

figure(fig_number); clf;
pos = get(fig_number,'Position');

[n_row,n_col] = size(cell_list);

fig_size_x = x_size*n_col+(n_col+1)*gap_x;
fig_size_y = y_size*n_row+(n_row+1)*gap_x;

set(fig_number,'Units','points', ...
                 'BackingStore','off', ...
                 'Color',[0.8 0.8 0.8], ...
                 'MenuBar','none', ...
                 'Resize','off', ...
                 'Name',title_figure, ...
                 'Position',[pos(1) pos(2) fig_size_x fig_size_y], ...
                 'NumberTitle','off'); %,'WindowButtonMotionFcn',['figure(' num2str(fig_number)
');']);

h_mat = zeros(n_row,n_col);

posx = zeros(n_row,n_col);
posy = zeros(n_row,n_col);

for i=n_row:-1:1,
  for j = n_col:-1:1,
    posx(i,j) = gap_x+(j-1)*(x_size+gap_x);
    posy(i,j) = fig_size_y - i*(gap_x+y_size);
  end;
end;

%disp('ok');
```

```
for i=n_row:-1:1,
   for j = n_col:-1:1,
      if ~isempty(cell_list{i,j}),
         if ~isempty(cell_list{i,j}{1}) & ~isempty(cell_list{i,j}{2}),
            h_mat(i,j) = uicontrol('Parent',fig_number, ...
               'Units','points', ...
               'Callback',cell_list{i,j}{2}, ...
               'ListboxTop',0, ...
               'Position',[posx(i,j)  posy(i,j)  x_size   y_size], ...
               'String',cell_list{i,j}{1}, ...
               'fontsize',font_size,...
               'fontname',font_name,...
               'Tag','Pushbutton1');
         end;
      end;
   end;
end;
```

A.2 Go_omni_calib (go_omni_calib_itr_s.m)

```
% This program is free software; you can redistribute it and/or
% modify it under the terms of the GNU General Public License
% as published by the Free Software Foundation, version 2.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program; if not, write to the Free Software Foundation,
% Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               %
%     Minimisation function            %
%     Modified by David Stone 4/2014   %
%                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Main calibration function.
% Computes the intrinsic and extrinsic parameters.
%
% Input: see "click_clib.m"
%
```

```
% Output:
%   adds variables to paramEst structure and modifies estimates
%   for the extrinsic parameters :
%     gammac: Camera focal length
%     cc : Principal point coordinates
%     alpha_c : Skew coefficient
%     kc : Distortion coefficients
%     KK : The camera matrix (containing gammac and cc)
%     Tw : list of extrinsic translation parameters
%     Qw : list of extrinsic rotation parameters
%     y  : point reprojections
%     ex : list of reprojection errors
%
% Method: Uses the LevenbergMarquardt algorithm to minimise the
%        reprojection error in the least squares sense over the intrinsic
%        camera parameters, and the extrinsic parameters (3D locations of the grids in space)
%
% Note: If the intrinsic camera parameters (gammac, cc, kc)
%       where initialised thanks to the mirror border extraction.
%
% Note: The row vector active_images consists of zeros and ones. To deactivate an image, set
the
%      corresponding entry in the active_images vector to zero.
%

function [paramEst, images] =
go_omni_calib_optim_iter_s(minInfo,images,gen_KK_est,gridInfo,paramEst)
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Minimisation properties
%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~exist('recompute_extrinsic'),
  recompute_extrinsic = 1; % Set this variable to 0 in case you do
                         % not want to recompute the extrinsic parameters
                                  % at each iteration.
end

if ~exist('check_cond'),
  check_cond = 1; % Set this variable to 0 in case you don't want to extract view dynamically
end
counterr = 0;
errf = [];
err = [];
stt = [];

persistent errt
    if isempty(errt)
```

```
        errt = [];
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Parameters to estimate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Variables :
% [Qw Tw Dist alpha gamma c];
if ~isfield(images,'desactivated_images')
  images.desactivated_images = [];
end
if ~isfield(paramEst,'est_dist')
  % Estimate distortion
  paramEst.est_dist = [1;1;1;1;0];
end
if ~isfield(paramEst,'est_alpha')
  % By default, do not estimate skew
  paramEst.est_alpha = 0;
end
if ~isfield(paramEst,'est_gammac')
  % Set to zero if you do not want to estimate
  % the combined focal length
  paramEst.est_gammac = [1;1];
end
if ~isfield(paramEst,'est_aspect_ratio')
  % Aspect ratio
  paramEst.est_aspect_ratio = 1;
end
if ~isfield(paramEst,'center_optim')
  % Set this variable to 0 if your do
  % not want to estimate the principal point
  paramEst.center_optim = 1;
end

est_xi = paramEst.est_xi;
est_dist = paramEst.est_dist;
est_alpha = paramEst.est_alpha;
est_gammac = paramEst.est_gammac;
est_aspect_ratio = paramEst.est_aspect_ratio;
center_optim = paramEst.center_optim;

nx = images.nx;
ny = images.ny;
n_ima = images.n_ima;

active_images = images.active_images;
ind_active = find(images.active_images);
```

```
% Load variables
xi = paramEst.xi;
if isfield(paramEst,'kc')
  kc = paramEst.kc;
end
if isfield(paramEst,'alpha_c')
  alpha_c = paramEst.alpha_c;
end
if isfield(paramEst,'gammac')
  gammac = paramEst.gammac;
end
if isfield(paramEst,'cc')
  cc = paramEst.cc;
end

% A quick fix for solving conflict
if ~isequal(est_gammac,[1;1]),
  est_aspect_ratio=1;
end
if ~est_aspect_ratio,
  est_gammac=[1;1];
end

if est_xi
  fprintf(1,'Xi will be estimated (est_xi = 1).\n');
else
  fprintf(1,'Xi will not be estimated (est_xi = 0).\n');
end

if ~est_aspect_ratio,
    fprintf(1,'Aspect ratio not optimized (est_aspect_ratio = 0) -> gammac(1)=gammac(2). Set
est_aspect_ratio to 1 for estimating aspect ratio.\n');
else
  if isequal(est_gammac,[1;1]),
    fprintf(1,'Aspect ratio optimized (est_aspect_ratio = 1) -> both components of gammac are
estimated (DEFAULT).\n');
  end
end

if ~isequal(est_gammac,[1;1]),
  if isequal(est_gammac,[1;0]),
    fprintf(1,'The first component of focal (gammac(1)) is estimated, but not the second one
(est_gammac=[1;0])\n');
  else
    if isequal(est_gammac,[0;1]),
   fprintf(1,'The second component of focal (gammac(1)) is estimated, but not the first one (est_gammac=[0;1])\n');
  else
   fprintf(1,'The focal vector gammac is not optimized (est_gammac=[0;0])\n');
  end
 end
end

if ~center_optim, % In the case where the principal point is not estimated, keep it at the center of the image
 fprintf(1,'Principal point not optimized (center_optim=0). ');
 if ~exist('cc'),
  fprintf(1,'It is kept at the center of the image.\n');
  cc = [(nx-1)/2;(ny-1)/2];
 else
  fprintf(1,'Note: to set it in the middle of the image, clear variable cc, and run calibration again.\n');
 end
else
  fprintf(1,'Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0\n');
end

if ~center_optim & (est_alpha),
 fprintf(1,'WARNING: Since there is no principal point estimation (center_optim=0), no skew estimation (est_alpha = 0)\n');
 est_alpha = 0;
end

if ~est_alpha,
 fprintf(1,'Skew not optimized (est_alpha=0) - (DEFAULT)\n');
 alpha_c = 0;
else
 fprintf(1,'Skew optimized (est_alpha=1). To disable skew estimation, set est_alpha=0.\n');
end

if ~prod(double(est_dist))&exist('kc')
 % If no distortion estimated, set to
 % zero the variables that are not estimated
 kc = kc .* est_dist;
end

if ~prod(double(est_gammac)),
 fprintf(1,'Warning: The focal length is not fully estimated (est_gammac ~= [1;1])\n');
end
```

```
% Put the initial estimates in param
if exist('gammac')
  if ~est_aspect_ratio
    gammac(1) = (gammac(1)+gammac(2))/2;
    gammac(2) = gammac(1);
  end
  XI = [xi;kc;alpha_c;gammac;cc];
else
  gammac = [gen_KK_est(1,1);gen_KK_est(2,2)];
  cc = [gen_KK_est(1:2,3)];
  if ~est_aspect_ratio
    gammac(1) = (gammac(1)+gammac(2))/2;
    gammac(2) = gammac(1);
  end
  % Initialise the distortions with 0 and the other values with
  % the estimation using the mirror border
  XI = [xi;zeros(5,1);0;gammac;cc];
end

%XI

param = [XI;zeros(7*n_ima,1)];

for kk = ind_active
  if isempty(paramEst.Qw{kk})
    fprintf(1,'Extrinsic parameters at frame %d do not exist\n',kk);
    return
  end
  param(11+7*(kk-1) + 1:11+7*(kk-1) + 7) = [paramEst.Qw{kk};paramEst.Tw{kk}];
end

%-------------------- Main Optimization:

fprintf(1,['\nMain calibration optimization procedure - Number of' ...
           ' images : %d\n'], length(ind_active));

fprintf(1,'Gradient descent iterations : ');

xi = XI(1);
gammac = XI(8:9);
cc = XI(10:11);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Optimisation settings %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
iter = 0;
```

```
cond_thresh = 1e-30;

VERS = version;
VERS =  VERS(1);

if(VERS=='7')
  disp(['WARNING: removing singular matrix warning and managing it' ...
          ' internally.'])
  warning('off','MATLAB:nearlySingularMatrix')
end

emax1 = 1e-10;

taux = minInfo.taux;
nu = minInfo.nu;
MaxIterBiased = minInfo.MaxIterBiased;
recompute_extrinsic_biased = minInfo.recompute_extrinsic_biased;
freqRecompExtrBiased = minInfo.freqRecompExtrBiased;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
% The following vector helps to select the
% variables to update (for only active images):
selected_variables = [est_xi;est_dist;est_alpha;est_gammac;center_optim*ones(2,1);...
                      reshape(ones(7,1)*active_images,7*n_ima,1)];

if ~est_aspect_ratio
  if isequal(est_gammac,[1;1]) | isequal(est_gammac,[1;0])
    selected_variables(9) = 0;
  end
end

ind_Jac = find(selected_variables)';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%

[sfx,ex3,JJ3] = buildJacobian_s(n_ima, gridInfo, param, ind_active);
%JJ2_inv_old = inv(JJ3);
%JJ2_inv_old = pinv(JJ3);

mu = taux*max(max(JJ3));
found=(max(abs(ex3))<emax1);

do_recomp = 1;

while ~found&(iter<MaxIterBiased)
```

```
 fprintf(1,'%d...',iter+1);

 if (mu==Inf)|(mu==NaN)
   mu = 1;
 end
 JJ3 = JJ3+mu*eye(size(JJ3,1));

% if rcond(JJ3)<cond_thresh
%   disp('Matrix badly conditionned, stopping...')
%   break
% end

 if ~est_aspect_ratio & isequal(est_gammac,[1;1]),
   param(9) = param(8);
 end

 %size(JJ3)
 JJ3_old = JJ3;
 ex3_old = ex3;

 JJ3 = JJ3(ind_Jac,ind_Jac);
 ex3 = ex3(ind_Jac);

 %hlm = -inv(JJ3)*ex3;
 hlm = -pinv(JJ3)*ex3;

 param_old = param;

 param(ind_Jac) = param(ind_Jac)+hlm;
 sfxp1 = buildValue_s(n_ima, gridInfo, param, ind_active);
 sFx = norm(sfx)^2;
 sFxp1 = norm(sfxp1)^2;

 quote = (sFx-sFxp1)/(0.5*hlm'*(mu*hlm-ex3));

 if quote>0
   [sfx,ex3,JJ3] = buildJacobian_s(n_ima, gridInfo, param, ind_active);
   found=max(abs(ex3))<emax1;
   mu=mu*max(1/3,1-(2*quote-1)^3);
   nu=2;
   do_recomp = 1;
 else
   JJ3 = JJ3_old;
   ex3 = ex3_old;
   param = param_old;
   mu=mu*nu;
```

```
    nu=2*nu;
  end

  %% Second step: (optional) - It makes convergence faster, and the region of convergence
LARGER!!!
  %% Recompute the extrinsic parameters only using compute_extrinsic.m (this may be useful
sometimes)
  %% The complete gradient descent method is useful to precisely update the intrinsic
parameters.

  if recompute_extrinsic&(mod(iter+1,freqRecompExtrBiased)==0) %==0,
    if do_recomp
      do_recomp = 0;
      fprintf(1,'(r) ');

      for kk = ind_active

            Qw_current = param(11+7*(kk-1) + 1:11+7*(kk-1) + 4);
            Tw_current = param(11+7*(kk-1) + 5:11+7*(kk-1) + 7);

            xp = omniCamProjection_s(cell2mat(gridInfo.X{kk}),...
                                [Qw_current; Tw_current;param(1:11)]);

            error_init = mean(mean(abs(xp-cell2mat(gridInfo.x{kk})),2));

            [Qw_new,Tw_new,error,k] = fastOmniPnP(cell2mat(gridInfo.X{kk}),
cell2mat(gridInfo.x{kk}),...
                                            [Qw_current;Tw_current;param(1:11)]);

            %error
            xp = omniCamProjection(cell2mat(gridInfo.X{kk}),[Qw_new;Tw_new;param(1:11)]);
            error_new = mean(mean(abs(xp-cell2mat(gridInfo.x{kk})),2));

    if check_cond
              if error_new/error_init>5
                active_images(kk) = 0;
                fprintf(1,'\nWarning: View #%d is causing problems. This image is now set inactive.
(note: to disactivate this option, set check_cond=0)\n',kk);
%               deactivated_images = [deactivated_images kk];
                Qw_new = NaN*ones(4,1);
                Tw_new = NaN*ones(3,1);
                images.active_images = active_images;
              end
            end
            param(11+7*(kk-1) + 1:11+7*(kk-1) + 4) = Qw_new;
            param(11+7*(kk-1) + 5:11+7*(kk-1) + 7) = Tw_new;
      end
    end
    jj= iter;
    errf(jj)= error_new;

  end

  iter = iter + 1;

end

fprintf(1,'done\n');

%%%--------------------------- Computation of the error of estimation:

fprintf(1,'Estimation of uncertainties...');

%check_active_images;

solution = param;

% Extraction of the parameters for computing the right reprojection error:
paramEst.xi = solution(1);
paramEst.kc = solution(2:6);
paramEst.alpha_c = solution(7);
paramEst.gammac = solution(8:9);
paramEst.cc = solution(10:11);

for kk = ind_active
  %1:length(ind_active)
  %index = ind_active(kk);

  paramEst.Qw{kk} = solution(11+7*(kk-1) + 1: 11+7*(kk-1) + 4);
  paramEst.Tw{kk} = solution(11+7*(kk-1) + 5: 11+7*(kk-1) + 7);

end

% Recompute the error (in the vector ex):
[err_mean_abs,err_std_abs,err_std,paramEst] = ...
    comp_omni_error_s(images,gen_KK_est,paramEst,gridInfo);
%comp_omni_sphere_error;

[sfx,ex3,JJ3] = buildJacobian_s(n_ima, gridInfo, param, ind_active);
JJ3 = JJ3(ind_Jac,ind_Jac);
JJ3 = 0.001*isnan(JJ3);
```

```
sigma_x = std(sfx(:));

%param_error = 3*sqrt(full(diag(inv(JJ3))))*sigma_x;
param_error = 3*sqrt(full(diag(pinv(JJ3))))*sigma_x;

index_val = 1;

paramEst.xi_error = NaN;
paramEst.kc_error = NaN*ones(5,1);
paramEst.alpha_c_error = NaN;
paramEst.gammac_error = NaN*ones(2,1);
paramEst.cc_error = NaN*ones(2,1);

if est_xi
  paramEst.xi_error = param_error(1);
  index_val = index_val+1;
end

for i=1:5
  if est_dist(i)
    paramEst.kc_error(i) = param_error(index_val);
    index_val = index_val + 1;
  end
end
if est_alpha
  paramEst.alpha_c_error = param_error(index_val);
  index_val = index_val + 1;
end
for i=1:2
  if est_gammac(i)
    paramEst.gammac_error(i) = param_error(index_val);
    index_val = index_val + 1;
  end
end

if center_optim
  paramEst.cc_error = param_error(index_val:index_val+1);
  index_val = index_val + 2;
end

% fprintf(1,'\n Average reprojection error computed for each chessboard [pixels]:\n\n');
%
xp = omniCamProjection(cell2mat(gridInfo.X{kk}),[Qw_new;Tw_new;param(1:11)]);
%
stt = (mean(abs(xp-cell2mat(gridInfo.x{kk})),2));
            err =(mean(stt));
    stderr = std(stt);
%
% for i=1:images.n_ima
%    fprintf(' %3.4f ± %3.4f\n',err(i),stderr(i));
% end
% n_ima = images.n_ima;
%
% avg_er = mean(err(:));
% avg_err = ones(size(ima_proc))* avg_er;
%
% figure (2)
% plot(ima_proc, err(:), 'rd--');
% hold on
% plot(ima_proc, avg_err, 'k.-');
%
errt = [errt err];
figure (3); hold on; plot(errt);

  % Initialise the distortions with 0 and the other values with
  % the estimation using the mirror border
  XI = [xi;zeros(5,1);0;gammac;cc];

%XI

%param = [XI;zeros(7*n_ima,1)];

% for kk == ind_active
%   if isempty(paramEst.Qw{kk})
%     fprintf(1,'Extrinsic parameters at frame %d do not exist\n',kk);
%     return
%   end
%   param(11+7*(kk-1) + 1:11+7*(kk-1) + 7) = [paramEst.Qw{kk};paramEst.Tw{kk}];
% end

%-------------------- Main Optimization:

fprintf(1,['\nMain calibration optimization procedure - Number of' ...
             ' images : %d\n'], length(ind_active));

fprintf(1,'Gradient descent iterations : ');

xi = XI(1);
gammac = XI(8:9);
cc = XI(10:11);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% Optimisation settings %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
iter = 0;
cond_thresh = 1e-30;

VERS = version;
VERS =  VERS(1);

if(VERS=='7')
  disp(['WARNING: removing singular matrix warning and managing it' ...
          ' internally.'])
  warning('off','MATLAB:nearlySingularMatrix')
end

emax1 = 1e-10;

taux = minInfo.taux;
nu = minInfo.nu;
MaxIterBiased = minInfo.MaxIterBiased;
recompute_extrinsic_biased = minInfo.recompute_extrinsic_biased;
freqRecompExtrBiased = minInfo.freqRecompExtrBiased;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
% The following vector helps to select the
% variables to update (for only active images):
selected_variables = [est_xi;est_dist;est_alpha;est_gammac;center_optim*ones(2,1);...
                      reshape(ones(7,1)*active_images,7*n_ima,1)];

if ~est_aspect_ratio
  if isequal(est_gammac,[1;1]) | isequal(est_gammac,[1;0])
    selected_variables(9) = 0;
  end
end

ind_Jac = find(selected_variables)';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%

[sfx,ex3,JJ3] = buildJacobian_s(n_ima, gridInfo, param, ind_active);
%JJ2_inv_old = inv(JJ3);
%JJ2_inv_old = pinv(JJ3);

mu = taux*max(max(JJ3));
found=(max(abs(ex3))<emax1);
```

```
do_recomp = 1;

while ~found&(iter<MaxIterBiased)
  fprintf(1,'%d...',iter+1);

  if (mu==Inf)|(mu==NaN)
    mu = 1;
  end
  JJ3 = JJ3+mu*eye(size(JJ3,1));

%  if rcond(JJ3)<cond_thresh
%    disp('Matrix badly conditionned, stopping...')
%    break
% end

  if ~est_aspect_ratio & isequal(est_gammac,[1;1]),
    param(9) = param(8);
  end

   %size(JJ3)
  JJ3_old = JJ3;
  ex3_old = ex3;

  JJ3 = JJ3(ind_Jac,ind_Jac);
  ex3 = ex3(ind_Jac);

  %hlm = -inv(JJ3)*ex3;
  hlm = -pinv(JJ3)*ex3;

  param_old = param;

  param(ind_Jac) = param(ind_Jac)+hlm;
  sfxp1 = buildValue_s(n_ima, gridInfo, param, ind_active);
  sFx = norm(sfx)^2;
  sFxp1 = norm(sfxp1)^2;

  quote = (sFx-sFxp1)/(0.5*hlm'*(mu*hlm-ex3));

  if quote>0
    [sfx,ex3,JJ3] = buildJacobian_s(n_ima, gridInfo, param, ind_active);
    found=max(abs(ex3))<emax1;
    mu=mu*max(1/3,1-(2*quote-1)^3);
    nu=2;
    do_recomp = 1;
  else
    JJ3 = JJ3_old;
  ex3 = ex3_old;
  param = param_old;
  mu=mu*nu;
  nu=2*nu;
 end

 %% Second step: (optional) - It makes convergence faster, and the region of convergence
LARGER!!!
 %% Recompute the extrinsic parameters only using compute_extrinsic.m (this may be useful
sometimes)
 %% The complete gradient descent method is useful to precisely update the intrinsic
parameters.

 if recompute_extrinsic&(mod(iter+1,freqRecompExtrBiased)==0) %==0,
  if do_recomp
   do_recomp = 0;
   fprintf(1,'(r) ');

   for kk = ind_active

      Qw_current = param(11+7*(kk-1) + 1:11+7*(kk-1) + 4);
      Tw_current = param(11+7*(kk-1) + 5:11+7*(kk-1) + 7);

      xp = omniCamProjection_s(cell2mat(gridInfo.X{kk}),...
                      [Qw_current; Tw_current;param(1:11)]);

      error_init = mean(mean(abs(xp-cell2mat(gridInfo.x{kk})),2));

      [Qw_new,Tw_new,error,k] = fastOmniPnP(cell2mat(gridInfo.X{kk}),
cell2mat(gridInfo.x{kk}),...
                                  [Qw_current;Tw_current;param(1:11)]);

      %error
      xp = omniCamProjection(cell2mat(gridInfo.X{kk}),[Qw_new;Tw_new;param(1:11)]);
      error_new = mean(mean(abs(xp-cell2mat(gridInfo.x{kk})),2));

  if check_cond
       if error_new/error_init>5
        active_images(kk) = 0;
        fprintf(1,'\nWarning: View #%d is causing problems. This image is now set inactive.
(note: to disactivate this option, set check_cond=0)\n',kk);
%       deactivated_images = [deactivated_images kk];
        Qw_new = NaN*ones(4,1);
        Tw_new = NaN*ones(3,1);
        images.active_images = active_images;
       end
```

```
            end
            param(11+7*(kk-1) + 1:11+7*(kk-1) + 4) = Qw_new;
            param(11+7*(kk-1) + 5:11+7*(kk-1) + 7) = Tw_new;
      end
    end
    jj= iter;
    errf(jj)= error_new;

  end

  iter = iter + 1;

end

fprintf(1,'done\n');

%%%--------------------------- Computation of the error of estimation:

fprintf(1,'Estimation of uncertainties...');


%check_active_images;

solution = param;

% Extraction of the parameters for computing the right reprojection error:
paramEst.xi = solution(1);
paramEst.kc = solution(2:6);
paramEst.alpha_c = solution(7);
paramEst.gammac = solution(8:9);
paramEst.cc = solution(10:11);

for kk = ind_active
  %1:length(ind_active)
  %index = ind_active(kk);

  paramEst.Qw{kk} = solution(11+7*(kk-1) + 1: 11+7*(kk-1) + 4);
  paramEst.Tw{kk} = solution(11+7*(kk-1) + 5: 11+7*(kk-1) + 7);

end

% Recompute the error (in the vector ex):
[err_mean_abs,err_std_abs,err_std,paramEst] = ...
    comp_omni_error_s(images,gen_KK_est,paramEst,gridInfo);
%comp_omni_sphere_error;
```

```
[sfx,ex3,JJ3] = buildJacobian_s(n_ima, gridInfo, param, ind_active);
JJ3 = JJ3(ind_Jac,ind_Jac);
JJ3 = 0.001*isnan(JJ3);

sigma_x = std(sfx(:));

%param_error = 3*sqrt(full(diag(inv(JJ3))))*sigma_x;
param_error = 3*sqrt(full(diag(pinv(JJ3))))*sigma_x;

index_val = 1;

paramEst.xi_error = NaN;
paramEst.kc_error = NaN*ones(5,1);
paramEst.alpha_c_error = NaN;
paramEst.gammac_error = NaN*ones(2,1);
paramEst.cc_error = NaN*ones(2,1);

if est_xi
  paramEst.xi_error = param_error(1);
  index_val = index_val+1;
end

for i=1:5
  if est_dist(i)
    paramEst.kc_error(i) = param_error(index_val);
    index_val = index_val + 1;
  end
end
if est_alpha
  paramEst.alpha_c_error = param_error(index_val);
  index_val = index_val + 1;
end
for i=1:2
  if est_gammac(i)
    paramEst.gammac_error(i) = param_error(index_val);
    index_val = index_val + 1;
  end
end

if center_optim
  paramEst.cc_error = param_error(index_val:index_val+1);
  index_val = index_val + 2;
end

%fprintf(1,'\n Average reprojection error computed for each chessboard [pixels]:\n\n');
```

```
%xp = omniCamProjection(cell2mat(gridInfo.X{kk}),[Qw_new;Tw_new;param(1:11)]);

%stt = (mean(abs(xp-cell2mat(gridInfo.x{kk})),2));
%           err =(mean(stt));
 %   stderr = std(stt);

% for i=1:images.n_ima
%    fprintf(' %3.4f ± %3.4f\n',err(i),stderr(i));
%end
%n_ima = images.n_ima;

%avg_er = mean(err(:));
%avg_err = ones(size(ima_proc))* avg_er;

%figure (2)
%plot(ima_proc, err(:), 'rd--');
%hold on
%plot(ima_proc, avg_err, 'k.-');
%

fprintf(1,'done\n');

show_intrinsic(paramEst,err_mean_abs,err_std_abs);

%%% Some recommendations to the user to reject some of the difficult unkowns... Still in debug
mode.

alpha_c_min = paramEst.alpha_c - paramEst.alpha_c_error/2;
alpha_c_max = paramEst.alpha_c + paramEst.alpha_c_error/2;

if (alpha_c_min < 0) & (alpha_c_max > 0)
  fprintf(1,'Recommendation: The skew coefficient alpha_c is found to be equal to zero (within
its uncertainty).\n');
  fprintf(1,'              You may want to reject it from the optimization by setting
paramEst.est_alpha=0 and run Calibration\n\n');
end

kc_min = paramEst.kc - paramEst.kc_error/2;
kc_max = paramEst.kc + paramEst.kc_error/2;

prob_kc = (kc_min < 0) & (kc_max > 0);

if ~(prob_kc(3) & prob_kc(4))

  % Initialise the distortions with 0 and the other values with
  % the estimation using the mirror border
  XI = [xi;zeros(5,1);0;gammac;cc];
end

%XI

param = [XI;zeros(7*n_ima,1)];

for kk = ind_active
  if isempty(paramEst.Qw{kk})
    fprintf(1,'Extrinsic parameters at frame %d do not exist\n',kk);
    return
  end
  param(11+7*(kk-1) + 1:11+7*(kk-1) + 7) = [paramEst.Qw{kk};paramEst.Tw{kk}];
end

%-------------------- Main Optimization:

fprintf(1,['\nMain calibration optimization procedure - Number of' ...
              ' images : %d\n'], length(ind_active));

fprintf(1,'Gradient descent iterations : ');

xi = XI(1);
gammac = XI(8:9);
cc = XI(10:11);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Optimisation settings %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
iter = 0;
cond_thresh = 1e-30;

VERS = version;
VERS =  VERS(1);

if(VERS=='7')
  disp(['WARNING: removing singular matrix warning and managing it' ...
           ' internally.'])
  warning('off','MATLAB:nearlySingularMatrix')
end

emax1 = 1e-10;

taux = minInfo.taux;
nu = minInfo.nu;
MaxIterBiased = minInfo.MaxIterBiased;
```

```
recompute_extrinsic_biased = minInfo.recompute_extrinsic_biased;
freqRecompExtrBiased = minInfo.freqRecompExtrBiased;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
% The following vector helps to select the
% variables to update (for only active images):
selected_variables = [est_xi;est_dist;est_alpha;est_gammac;center_optim*ones(2,1);...
                      reshape(ones(7,1)*active_images,7*n_ima,1)];

if ~est_aspect_ratio
  if isequal(est_gammac,[1;1]) | isequal(est_gammac,[1;0])
    selected_variables(9) = 0;
  end
end

ind_Jac = find(selected_variables)';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%

[sfx,ex3,JJ3] = buildJacobian_s(n_ima, gridInfo, param, ind_active);
%JJ2_inv_old = inv(JJ3);
%JJ2_inv_old = pinv(JJ3);

mu = taux*max(max(JJ3));
found=(max(abs(ex3))<emax1);

do_recomp = 1;

while ~found&(iter<MaxIterBiased)
  fprintf(1,'%d...',iter+1);

  if (mu==Inf)|(mu==NaN)
    mu = 1;
  end
  JJ3 = JJ3+mu*eye(size(JJ3,1));

%  if rcond(JJ3)<cond_thresh
%    disp('Matrix badly conditionned, stopping...')
%    break
% end

  if ~est_aspect_ratio & isequal(est_gammac,[1;1]),
    param(9) = param(8);
  end
  %size(JJ3)
  JJ3_old = JJ3;
  ex3_old = ex3;

  JJ3 = JJ3(ind_Jac,ind_Jac);
  ex3 = ex3(ind_Jac);

  %hlm = -inv(JJ3)*ex3;
  hlm = -pinv(JJ3)*ex3;

  param_old = param;

  param(ind_Jac) = param(ind_Jac)+hlm;
  sfxp1 = buildValue_s(n_ima, gridInfo, param, ind_active);
  sFx = norm(sfx)^2;
  sFxp1 = norm(sfxp1)^2;

  quote = (sFx-sFxp1)/(0.5*hlm'*(mu*hlm-ex3));

  if quote>0
    [sfx,ex3,JJ3] = buildJacobian_s(n_ima, gridInfo, param, ind_active);
    found=max(abs(ex3))<emax1;
    mu=mu*max(1/3,1-(2*quote-1)^3);
    nu=2;
    do_recomp = 1;
  else
    JJ3 = JJ3_old;
    ex3 = ex3_old;
    param = param_old;
    mu=mu*nu;
    nu=2*nu;
  end

  %% Second step: (optional) - It makes convergence faster, and the region of convergence
LARGER!!!
  %% Recompute the extrinsic parameters only using compute_extrinsic.m (this may be useful
sometimes)
  %% The complete gradient descent method is useful to precisely update the intrinsic
parameters.

  if recompute_extrinsic&(mod(iter+1,freqRecompExtrBiased)==0) %==0,
    if do_recomp
      do_recomp = 0;
      fprintf(1,'(r) ');
      for kk = ind_active

            Qw_current = param(11+7*(kk-1) + 1:11+7*(kk-1) + 4);
            Tw_current = param(11+7*(kk-1) + 5:11+7*(kk-1) + 7);

            xp = omniCamProjection_s(cell2mat(gridInfo.X{kk}),...
                                           [Qw_current; Tw_current;param(1:11)]);

            error_init = mean(mean(abs(xp-cell2mat(gridInfo.x{kk})),2));

            [Qw_new,Tw_new,error,k] = fastOmniPnP(cell2mat(gridInfo.X{kk}),
cell2mat(gridInfo.x{kk}),...
                                                                  [Qw_current;Tw_current;param(1:11)]);

            %error
            xp = omniCamProjection(cell2mat(gridInfo.X{kk}),[Qw_new;Tw_new;param(1:11)]);
            error_new = mean(mean(abs(xp-cell2mat(gridInfo.x{kk})),2));

    if check_cond
              if error_new/error_init>5
                active_images(kk) = 0;
                fprintf(1,'\nWarning: View #%d is causing problems. This image is now set inactive.
(note: to disactivate this option, set check_cond=0)\n',kk);
%               deactivated_images = [deactivated_images kk];
                Qw_new = NaN*ones(4,1);
                Tw_new = NaN*ones(3,1);
                images.active_images = active_images;
              end
            end
            param(11+7*(kk-1) + 1:11+7*(kk-1) + 4) = Qw_new;
            param(11+7*(kk-1) + 5:11+7*(kk-1) + 7) = Tw_new;
      end
    end
    jj= iter;
    errf(jj)= error_new;

  end

  iter = iter + 1;

end

fprintf(1,'done\n');

%%%--------------------------- Computation of the error of estimation:
```

```
fprintf(1,'Estimation of uncertainties...');

%check_active_images;

solution = param;

% Extraction of the parameters for computing the right reprojection error:
paramEst.xi = solution(1);
paramEst.kc = solution(2:6);
paramEst.alpha_c = solution(7);
paramEst.gammac = solution(8:9);
paramEst.cc = solution(10:11);

for kk = ind_active
  %1:length(ind_active)
  %index = ind_active(kk);

  paramEst.Qw{kk} = solution(11+7*(kk-1) + 1: 11+7*(kk-1) + 4);
  paramEst.Tw{kk} = solution(11+7*(kk-1) + 5: 11+7*(kk-1) + 7);

end

% Recompute the error (in the vector ex):
[err_mean_abs,err_std_abs,err_std,paramEst] = ...
    comp_omni_error_s(images,gen_KK_est,paramEst,gridInfo);
%comp_omni_sphere_error;

[sfx,ex3,JJ3] = buildJacobian_s(n_ima, gridInfo, param, ind_active);
JJ3 = JJ3(ind_Jac,ind_Jac);
JJ3 = 0.001*isnan(JJ3);

sigma_x = std(sfx(:));

%param_error = 3*sqrt(full(diag(inv(JJ3))))*sigma_x;
param_error = 3*sqrt(full(diag(pinv(JJ3))))*sigma_x;

index_val = 1;

paramEst.xi_error = NaN;
paramEst.kc_error = NaN*ones(5,1);
paramEst.alpha_c_error = NaN;
paramEst.gammac_error = NaN*ones(2,1);
paramEst.cc_error = NaN*ones(2,1);

if est_xi
  paramEst.xi_error = param_error(1);
  index_val = index_val+1;
end

for i=1:5
  if est_dist(i)
    paramEst.kc_error(i) = param_error(index_val);
    index_val = index_val + 1;
  end
end
if est_alpha
  paramEst.alpha_c_error = param_error(index_val);
  index_val = index_val + 1;
end
for i=1:2
  if est_gammac(i)
    paramEst.gammac_error(i) = param_error(index_val);
    index_val = index_val + 1;
  end
end

if center_optim
  paramEst.cc_error = param_error(index_val:index_val+1);
  index_val = index_val + 2;
end

%fprintf(1,'\n Average reprojection error computed for each chessboard [pixels]:\n\n');

%xp = omniCamProjection(cell2mat(gridInfo.X{kk}),[Qw_new;Tw_new;param(1:11)]);

%stt = (mean(abs(xp-cell2mat(gridInfo.x{kk})),2));
%             err =(mean(stt));
 %   stderr = std(stt);

% for i=1:images.n_ima
%    fprintf(' %3.4f ± %3.4f\n',err(i),stderr(i));
%end
%n_ima = images.n_ima;

%avg_er = mean(err(:));
%avg_err = ones(size(ima_proc))* avg_er;

%figure (2)
%plot(ima_proc, err(:), 'rd--');
%hold on
%plot(ima_proc, avg_err, 'k.-');
%
```

```
fprintf(1,'done\n');

show_intrinsic(paramEst,err_mean_abs,err_std_abs);

%%% Some recommendations to the user to reject some of the difficult unkowns... Still in debug
mode.

alpha_c_min = paramEst.alpha_c - paramEst.alpha_c_error/2;
alpha_c_max = paramEst.alpha_c + paramEst.alpha_c_error/2;

if (alpha_c_min < 0) & (alpha_c_max > 0)
  fprintf(1,'Recommendation: The skew coefficient alpha_c is found to be equal to zero (within
its uncertainty).\n');
  fprintf(1,'               You may want to reject it from the optimization by setting
paramEst.est_alpha=0 and run Calibration\n\n');
end

kc_min = paramEst.kc - paramEst.kc_error/2;
kc_max = paramEst.kc + paramEst.kc_error/2;

prob_kc = (kc_min < 0) & (kc_max > 0);

if ~(prob_kc(3) & prob_kc(4))
fprintf(1,'done\n');
end
show_intrinsic(paramEst,err_mean_abs,err_std_abs);

%%% Some recommendations to the user to reject some of the difficult unkowns... Still in debug
mode.

alpha_c_min = paramEst.alpha_c - paramEst.alpha_c_error/2;
alpha_c_max = paramEst.alpha_c + paramEst.alpha_c_error/2;

if (alpha_c_min < 0) & (alpha_c_max > 0)
  fprintf(1,'Recommendation: The skew coefficient alpha_c is found to be equal to zero (within
its uncertainty).\n');
  fprintf(1,'               You may want to reject it from the optimization by setting
paramEst.est_alpha=0 and run Calibration\n\n');
end

kc_min = paramEst.kc - paramEst.kc_error/2;
kc_max = paramEst.kc + paramEst.kc_error/2;

prob_kc = (kc_min < 0) & (kc_max > 0);
```

```
if ~(prob_kc(3) & prob_kc(4))
  prob_kc(3:4) = [0;0];
end

if sum(prob_kc),
  fprintf(1,'Recommendation: Some distortion coefficients are found equal to zero (within their
uncertainties).\n');
  fprintf(1,'                To reject them from the optimization set
paramEst.est_dist=[%d;%d;%d;%d;%d] and run Calibration\n\n',est_dist & ~prob_kc);
end

return

%function value=gain(hlm,mu,g)
%value=1/2*hlm'*(mu*hlm-g);
```

A.3 Spherical Coordinates (Spherical_s.m)

```
function [theta phi r] = rect2sphere(X_avg,Y_avg,z,n_ima)
%
% Routine written by David L. Stone
%
%   Convert rectangular x, y, z coordinates to spherical
%   R, theta, phi
%

r = sqrt(X_avg .^2 + Y_avg .^2);
theta = atand(X_avg ./ Y_avg);
phi = atand(r/z);

for i = 1:n_ima
    if X_avg(1,i)  >= 0 && Y_avg(1,i) >= 0
        theta(1,i) = theta(1,i);
    elseif X_avg(1,i)  > 0 && Y_avg(1,i) < 0
        theta(1,i) = theta(1,i) + 180;
    elseif X_avg(1,i)  < 0 && Y_avg(1,i) < 0
        theta(1,i) = theta(1,i) + 180;
    elseif X_avg(1,i)  < 0 && Y_avg(1,i) > 0
        theta(1,i) = theta(1,i) + 360;
    end

end
% Written by David Stone 4/2014
```

```
n_ima = images.n_ima;
X_avg = zeros(n_ima);
Y_avg = zeros(n_ima);

%    Modified by David Stone Apr 2014
%xc1 = 241;
%yc1 = 324;

z=1.0;

x_avg = (gridInfo.x_mean - gen_KK_est(1,3));
y_avg = -(gridInfo.y_mean - gen_KK_est(2,3));

[theta_mean phi_mean r_mean] = rect2sphere(x_avg,y_avg,z,n_ima);

if isfield(paramEst,'y')
param_x = paramEst.y_mean(1,:);
param_y = paramEst.y_mean(2,:);

x_center = num2cell(gen_KK_est(1,3)*[ones(1,n_ima)]);
y_center = num2cell(gen_KK_est(2,3)*[ones(1,n_ima)]);

%x_center = mat2cell(x_center_tmp, 1, n_ima);
%y_center = mat2cell(y_center_tmp, 1, n_ima);

end

x_avg_dsc = cellfun(@minus, param_x,x_center );
y_avg_dsc = -cellfun(@minus, param_y,y_center );

[theta_dsc phi_dsc r_dsc] = rect2sphere(x_avg_dsc,y_avg_dsc,z,n_ima);
%r = 0.37 .*r;

%x_avg_mei = cellfun(@minus, param_x_mean,x_center );
%y_avg_mei = -cellfun(@minus, param_y_mean,y_center );

%[theta_mei phi_mei r_mei] = rect2sphere(x_avg_mei,y_avg_mei,z,n_ima);
%r = 0.37 .*r;

%[theta_abs phi_abs r_abs] = rect2sphere(X_abs_avg,Y_abs_avg,z,n_ima);
%r_abs = 0.37 .*r_abs;

cal_data_out;

yes_ready = input('are you ready? [] yes, other no ');
if isempty(yes_ready);
```

```
    readmeasured;
else
end
```

A.4 Multi-objective Optimization ()

```
function f = simple_multiobj2(x,a,b,c)
x = x(:); a = a(:); b = b(:); c = c(:); % all column vectors
f(1) = sqrt(1+norm(x-a)^2);
f(2) = 0.5*sqrt(1+norm(x-b)^2) + 2;
f(3) = 0.25*sqrt(1+norm(x-c)^2) - 4;

%Then run this:

a = zeros(2,1);
b = [2;1];
c = [3;-.5];
fun = @(u)simple_multiobj2(u,a,b,c);
[x,f,ef] = gamultiobj(fun,2)
scatter3(f(:,1),f(:,2),f(:,3),'k.');

%how to use the scatteredinterpolant

F = scatteredInterpolant(f(:,1),f(:,2),f(:,3),'linear','none');
sgr = min(f(:,1)):.01:max(f(:,1));
ygr = min(f(:,2)):.01:max(f(:,2));
[XX,YY] = meshgrid(sgr,ygr);
ZZ = F(XX,YY);
surf(XX,YY,ZZ,'LineStyle','none')
```

A.5 Compute Omnidirectional Error (comp_omni_error_s.m)

```
% This program is free software; you can redistribute it and/or
% modify it under the terms of the GNU General Public License
% as published by the Free Software Foundation, version 2.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program; if not, write to the Free Software Foundation,
% Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Recomputes the reprojection error         %
%      (based on version by JYB)              %
%                                    %
%   Created : 2005 (mod 11/03/06)             %
%    Author : Christopher Mei                 %
%    Modified by David Stone April 2014       %
%                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

% Input :
%   see "click_calib.m"
%
% Output :
%   err_mean_abs : mean absolute error
%   err_std_abs : absolute standard deviation
%   err_std : standard deviation
%

function [err_mean_abs,err_std_abs,err_std,paramEst] =
comp_omni_error_s(images,gen_KK_est,paramEst,gridInfo)

% Reproject the patterns on the images, and compute the pixel errors:
ex = []; % Global error vector

if ~isfield(paramEst,'gammac')
  XI=[paramEst.xi;zeros(5,1);0;...
      gen_KK_est(1,1);gen_KK_est(2,2); ...
      gen_KK_est(1,3);gen_KK_est(2,3)];
else
  XI=[paramEst.xi;paramEst.kc;paramEst.alpha_c;...
      paramEst.gammac;paramEst.cc];
end

active_images = images.active_images;
ind_active = find(active_images);

%for kk = 1:length(ind_active)
%  index = ind_active(kk);

%  if active_images(kk) & size(paramEst.Qw,2)>=kk & ~isempty(gridInfo.X{index})
for index=ind_active
  V = [paramEst.Qw{index};paramEst.Tw{index};XI];
  xp = omniCamProjection_s(gridInfo.X{index}, V);

  part_ex = xp-cell2mat(gridInfo.x{index});

  paramEst.y{index} = xp;
  paramEst.y_mean{1,index} = mean(xp(1,:));
  paramEst.y_mean{2,index} = mean(xp(2,:));
  paramEst.ex{index} = part_ex;

  ex=[ex part_ex];
end
%  end
```

A.6 Analyse Error (analyse_err_s.m)

```
% Modified version of 'analyse_error' from JYB
% Modified by Dave Stone
% The errors in X and Y of the pixel projections are plotted
% to identify incorrect extractions.
%
% Using arrays and no longer X_i, x_i, ... values
%

function analyse_error_s(images,gridInfo,paramEst)

biased_calib = 1;

if ~isfield(paramEst,'gammac')
  fprintf(1,'You should start by calibrating the sensor.');
  return
end

n_ima = images.n_ima;
ex = paramEst.ex;
y = paramEst.y;
active_images = images.active_images;

if n_ima ~=0,
  if biased_calib
    if ~exist('ex')
      fprintf(1,['Need to calibrate before analysing reprojection' ...
                 ' error or load a Calib_Results.mat file.\n']);
      return
    end
```

```
 else
  if ~exist('ex_sphere_pix_'),
   fprintf(1,['Need to calibrate before analysing reprojection' ...
            ' error or load a Calib_Results.mat file.\n']);
   return
  end
 end
end

if ~exist('no_grid'),
 no_grid = 0;
end

colors = 'brgkcm';

figure(5);

for kk = 1:n_ima,
 if images.active_images(kk)&exist('y')&~isempty(y{kk})
  if active_images(kk) & ~isnan(y{kk}(1,1)),

   if ~no_grid,
      XX_kk = gridInfo.X{kk};
      N_kk = size(XX_kk,2);

      if ~isempty(gridInfo.n_sq_x{kk})
       no_grid = 1;
      end

      if ~no_grid
       n_sq_x = gridInfo.n_sq_x{kk};
       n_sq_y = gridInfo.n_sq_y{kk};
       if (N_kk ~= ((n_sq_x+1)*(n_sq_y+1))),
        no_grid = 1;
       end
      end
   end

   if biased_calib
      plot(ex{kk}(1,:)',ex{kk}(2,:)',[colors(rem(kk-1,6)+1) '+'],'LineWidth',3,...
  'MarkerSize',15);
   else
      plot(exsphere_pix{kk}(1,:)',exsphere_pix{kk}(2,:)', ...
        [colors(rem(kk-1,6)+1) '+'],'LineWidth',3,...
  'MarkerSize',15);
   end
```

```
      hold on;
    end
  end
end

hold off;
axis('equal');
if 1, %~no_grid,
  title('Reprojection error (in pixel) - To exit: right button');
else
  title('Reprojection error (in pixel)');
end
xlabel('Error in x (pixels)','fontsize',24,'fontweight','b');
ylabel('Error in y (pixels)','fontsize',24,'fontweight','b');

set(5,'color',[1 1 1]);
set(5,'Name','error','NumberTitle','off');

ex_mat = cell2mat(ex);

if n_ima == 0,
  text(.5,.5,'No image data available','fontsize',24,'horizontalalignment' ,'center');
else
  if biased_calib
    err_std = std(ex_mat')';
    fprintf(1,'Pixel error:          err = [ %3.6f  ] (all active images)\n\n',err_std);
  else
    err_std_sphere_pix = std(ex_sphere_pix')';
    fprintf(1,'Pixel error:          err = [ %3.6f  ] (all active images)\n\n',err_std_sphere_pix);
  end

  b = 1;

  while b==1,

    [xp,yp,b] = ginput3(1);

    if b==1,
      if biased_calib
        ddd = (ex_mat(1,:)-xp).^2 + (ex_mat(2,:)-yp).^2;
      else
        ddd = (ex_sphere_pix(1,:)-xp).^2 + (ex_sphere_pix(2,:)-yp).^2;
      end

      [mind,indmin] = min(ddd);
    done = 0;
    kk_ima = 1;
    while (~done)&(kk_ima<=n_ima),
        %fprintf(1,'%d...',kk_ima);

        if ~images.active_images(kk_ima)
          kk_ima = kk_ima + 1;
          continue;
        end

        if biased_calib
          ex_kk = ex{kk_ima};
          sol_kk = find((ex_kk(1,:) == ex_mat(1,indmin))&(ex_kk(2,:) == ex_mat(2,indmin)));
        else
          ex_kk = exsphere_pix{kk_ima};
          sol_kk = find((ex_kk(1,:) == exsphere_pix(1,indmin))&(ex_kk(2,:) ==
exsphere_pix(2,indmin)));
        end

        if isempty(sol_kk),
          kk_ima = kk_ima + 1;
        else
          done = 1;
        end
    end

    xkk = gridInfo.x{kk_ima};
    xpt = xkk(:,sol_kk);

    if ~no_grid

        n_sq_x = gridInfo.n_sq_x{kk_ima};
        n_sq_y = gridInfo.n_sq_y{kk_ima};

        Nx = n_sq_x+1;
        Ny = n_sq_y+1;

        y1 = floor((sol_kk-1)./Nx);
        x1 = sol_kk - 1 - Nx*y1; %rem(sol_kk-1,Nx);

        y1 = (n_sq_y+1) - y1;
        x1 = x1 + 1;

        fprintf(1,'\n');
```

```
            fprintf(1,'Selected image: %d\n',kk_ima);
            fprintf(1,'Selected point index: %d\n',sol_kk);
            fprintf(1,'Pattern coordinates (in units of (dX,dY)): (X,Y)=(%d,%d)\n',[x1-1 y1-1]);
            fprintf(1,['Image coordinates (in pixel): (%3.2f,' ...
                        ' %3.2f)\n'],[xpt']);
            if biased_calib
              fprintf(1,'Pixel error = (%3.5f,%3.5f)\n',[ex_mat(1,indmin) ex_mat(2,indmin)]);
            else
              fprintf(1,'Pixel error = (%3.5f,%3.5f)\n', ...
                       [ex_sphere_pix(1,indmin) ex_sphere_pix(2,indmin)]);
            end
        else

            fprintf(1,'\n');
            fprintf(1,'Selected image: %d\n',kk_ima);
            fprintf(1,'Selected point index: %d\n',sol_kk);
            fprintf(1,'Image coordinates (in pixel): (%3.2f,%3.2f)\n',[xpt']);

            if biased_calib
              fprintf(1,'Pixel error = (%3.5f,%3.5f)\n',[ex_mat(1,indmin) ex_mat(2,indmin)]);
            else
              fprintf(1,'Pixel error = (%3.5f,%3.5f)\n',[ex_sphere_pix(1,indmin)
ex_sphere_pix(2,indmin)]);
            end
        end


        if
isfield(gridInfo,'wintx')&~isempty(gridInfo.wintx{kk_ima})&~isnan(gridInfo.wintx{kk_ima})

            wintx = gridInfo.wintx{kk_ima};
            winty = gridInfo.winty{kk_ima};

            fprintf(1,'Window size: (wintx,winty) = (%d,%d)\n',[wintx winty]);
        end

    end

  end

fprintf(1,'\n Average reprojection error computed for each chessboard [pixels]:\n\n');

err = sqrt(ex_mat(1,:).^2 + ex_mat(2,:).^2);
stderr = std(err(:));
```

```
for i=1:n_ima
   fprintf(' %3.6f ± %3.6f\n',err(i),stderr);
end

avg_er = mean(err(:));
avg_err = ones(1,47)* avg_er;

figure (6)
plot([1:n_ima], err(:), 'gs--','LineWidth',3,...
   'MarkerSize',15);
hold on
plot(1:n_ima, avg_err(:), 'k-','LineWidth',2);
xlabel('Calibration Boards','fontsize',24,'fontweight','b');
ylabel('Error in Pixels','fontsize',24,'fontweight','b');

set(6,'Name','error plot','NumberTitle','off');

hold off

disp('done');
```

The Chapter 3 source codes for MNDVI and TRF are as follows:

A.7 Vegetation Detection IR Region (vegdetect_IRR.m)
A.8 IR Region (IR_Region.m)
A.9 Genetic Algorithm Clustering (GACluster.m)
A.10 Clustering Cost Function (ClusteringCost2.m)

A.7 Vegetation Detection IR Region (vegdetect_IRR.m)

```
% 5/9/11 %% Import All Images
% Modified by Dave Stone to add IR region growing, NDVI with blue vice red,
% and visual / IR fusion  5/11/11 - 5/19/18
% Also added error handling

clc;    % Clear the command window.
close all;  % Close all figures (except those of imtool.)
imtool close all;  % Close all imtool figures.
clear;  % Erase all existing variables.
workspace;  % Make sure the workspace panel is showing.
```

```
fontSize = 14;

tic
link1='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\Veg\';
link2='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\IRimage\';
link3='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputRegion4Grow_2\';
link4='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\ODData\';
%link3='E:\VCU\Thesis\Transaction_Paper\PlantData\';
output1='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\outputODK\'; %
Output NDVIR
output2='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\outputODK2\'; %
Output  NDVIB
output3='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\outputODK3\'; %
Output NDVIRB
output4='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\outputODK4\'; %
Output NDVIR_B
output5='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\outputODK5\'; %
Output PVI
output6='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\outputODK6\'; %
Output DVI
%output4='E:\VCU\Thesis\Transaction_Paper\PlantData\PlantDataOut';
outputAnnotateB_CDF='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateB\CDF\';
outputAnnotateB_Hist='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateB\Hist\';
outputAnnotateR_CDF='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateR\CDF\';
outputAnnotateR_Hist='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateR\Hist\';
outputAnnotateRB_CDF='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateRB\CDF\';
outputAnnotateRB_Hist='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateRB\Hist\';
outputAnnotateR_B_CDF='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateR_B\CDF\';
outputAnnotateR_B_Hist='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateR_B\Hist\';
outputAnnotateP_CDF='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateP\CDF\';
outputAnnotateP_Hist='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateP\Hist\';
outputAnnotateD_CDF='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateD\CDF\';
outputAnnotateD_Hist='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\outputAnnotateD\Hist\';
```

```
kk=0;
% try       % Eror capture for image file not found
%     vis = dir('link1');
%  catch exception
%     fprintf('image file not found')
% end
%      num1 = length(vis);
% imC = cell(1, num1);
%
% for k = 1:num1
%     imC{k} = imread(vis(k).name);
%
% end
% try
% infrared = dir('link2');
% catch exception
%     fprintf('IR image file not found')
% end
% try
% num2 = length(infrared);
% catch exception
%     if num2 == 0
%         fprint(' infrared index 0')
%     end
% end
% irC = cell(1, num2);
% for k = 1:num2
%     string = ['ir/', infrared(k).name ];
%     irC{k} = rgb2gray(imread(string));
% end
%
% %% Mass Vegetation Detection
% veg = cell(1, num1);
% bin = cell(1, num1);
% T = 0.5;
% for k=1:num1
%     [veg{k}, bin{k}] = massVeg(imC{k},irC{k},T);
%     vegname = (['veg\veg',num2str(k),'.png']);
%     imwrite(veg{k},vegname);
%     binname = (['bin\bin',num2str(k),'.png']);
%     imwrite(bin{k},binname);
% end
%% Vegetation Detection of One Image
starting = 1;
maxODim = 20;
MNDVIArea = nan(1, maxODim);
```

```
trfArea = nan(1, maxODim);
imgArea = nan(1, maxODim);
vegGTarea = nan(1, maxODim);
mndviFP = nan(1, maxODim);
trfFP = nan(1, maxODim);


TData(maxODim,4) = zeros;
tLevel(maxODim,1) = zeros;
% Import Images

for i=starting:maxODim

%im = double(imread('crouch_behind_bush.jpg'));
%ir = double(rgb2gray(imread('ir/IRcrouch_behind_busha.png')));

% Read im and ir from veg and ir directories in Vegetation Data
im = imread(strcat(link1,'IMin', num2str(i),'.jpg'));
ir = imread(strcat(link2,'IRin', num2str(i),'.jpg'));
irrg = imread(strcat(link3,'IRRG', num2str(i),'.jpg'));
vegGT = imread(strcat(output1,'vegGT', num2str(i),'.jpg'));
imOD = imread(strcat(link4,'IM_OD', num2str(i),'.jpg'));
irOD = imread(strcat(link4,'IR_OD', num2str(i),'.jpg'));

irrg =(rgb2gray(irrg));
irrg = imbinarize(irrg);

% Read im and ir from OD Plant added by David L. Stone  12/11/17
%im = imread(strcat(link3,'Dataset1\RGB_OD_vegetation', num2str(i),'.jpg'));
%ir = imread(strcat(link3,'Dataset1\IR_OD_vegetation', num2str(i),'.jpg'));
sizeir = size(ir);
numrows = sizeir(1);
numcols = sizeir(2);
im = imresize(im,[numrows numcols]);

% preallocate for speed

T = 0.9;
l = 12;
% Write out the ir image to file
imwrite(ir,(strcat(output2,'ir', num2str(i),'.jpg')));
%imwrite(ir,(strcat(output3,'ir', num2str(i),'.jpg')));
% Compare IR to R with vegetation index
[biVeg] = irCompare(im,ir,T,'NDVI');
%[biVeg] = irCompare(im,ir,T,'NDVIB');  %Added by David L. Stone 5/16/18
% Flatten colors
```

```
[imFlat] = imageFlatten(uint8(im),l);
% Select Green and Brown
[biColor,gr,br] = colorCompare(im);
[biColorFlat,grFlat,brFlat] = colorCompare(imFlat);

% Proform morphological opperations
[biMorph1] = morphOps(~biVeg,5,2);
[biMorph2] = morphOps(~biColor,5,2);
[biMorph3] = morphOps(biColorFlat,5,2);

% Vegetation Highlighting
biTotal1 = biColor | biVeg;
biTotal2 = biColorFlat | biVeg;
[biMorph4] = morphOps(~biTotal1,5,2);

% Find Thermal Regions
%J = IR_region(rgb3gray(ir));
[T1, T2, level, Iavg, thresholdvalue, binaryImage]= VThreshold(ir);
J = irrg;

%J = IR_region(ir, T1, T2);
% Write out the threshold data
TData(i,:) = [T1, T2, thresholdvalue, Iavg];
tLevel(i,:) = [level];



veg_final = and(biTotal1, J(:,:,1));
veg_final1 = and(biTotal1, not(J(:,:,1)));
%veg_final = and(biTotal1, J(:,:));
%veg_final1 = and(biTotal1, not(J(:,:)));

% Calculate the various areas

imgsize = size (im);
imgarea = imgsize(1) * imgsize(2);

imgArea(i) = imgarea;
vegGTarea(i) = nnz(imbinarize(rgb2gray(vegGT)));
MNDVIArea(i) = nnz(biColor);

trfArea(i) = nnz(veg_final);

% Calculate the MNDVI false positives

mndvifp = biColor -(imbinarize(rgb2gray(vegGT)));
```

```
if mndvifp < 0
   mndvifp = 0;
end

mndviFP(i) = nnz(imbinarize(mndvifp));

% Calculate the MNDVI false positives

trffp = veg_final -(imbinarize(rgb2gray(vegGT)));

if trffp < 0
   trffp = 0;
end

trfFP(i) = nnz(imbinarize(trffp));

% Construct identification images addid by David L. Stone
[veg1] = vegHiglight(im,biVeg);
[veg2] = vegHiglight(im,biColor); %MNDVI image
[veg3] = vegHiglight(im,biColorFlat);
[veg4] = vegHiglight(im,veg_final); %MNDVI merged with J region growing
[veg5] = vegHiglight(im,veg_final1);
[veg6] = vegHiglight(im,biMorph1);
[veg7] = vegHiglight(im,biMorph2);
[veg8] = vegHiglight(im,biMorph3);
[veg9] = vegHiglight(im,biMorph4);

% Plot the color histogram for each image added by David L. Stone
   %Split into channels
Red = im(:,:,1);
Green = im(:,:,2);
Blue = im(:,:,3);

rcounts = histcounts(Red);
rCDF = cumsum(rcounts)/sum(rcounts);

gcounts = histcounts(Green);
gCDF = cumsum(gcounts)/sum(gcounts);

bcounts = histcounts(Blue);
bCDF = cumsum(bcounts)/sum(bcounts);

xrCDF = 1:1:length(rCDF);
xgCDF = 1:1:length(gCDF);
xbCDF = 1:1:length(bCDF);
```

```
%Get histValues for each channel
[yRed, x] = imhist(Red);
[yGreen, x] = imhist(Green);
[yBlue, x] = imhist(Blue);
xmax = length(x);
maxYR = max(yRed);
maxYG = max(yGreen);
maxYB = max(yBlue);
ymax = max([maxYR maxYG maxYB])+ 100;
% Plot the full histogram

nBins = 256;



rHist = imhist(im(:,:,1), nBins);
gHist = imhist(im(:,:,2), nBins);
bHist = imhist(im(:,:,3), nBins);


hFig1(i) = figure(kk+1);
% plot(xrCDF, rCDF, 'Red',  xgCDF, gCDF, 'Green', xbCDF, bCDF, 'Blue');
% saveas(gcf, (strcat(outputAnnotateR_CDF,'CDF-Im', num2str(i),'.fig')));

% subplot(2,1,1)
% imshow(mndvifp);
% title('MNDVI FP');
%
% subplot(2,1,2);
% imshow(trffp);
% title('TRF FP');


kk = kk + 1;
%RGBHIST Histogram Plot yRed, 'Red',
% hFig2(i) = figure(kk+2);
% kk = kk + 2;

% subplot(5,1,1);
% imshow(im);
% title('Input Image');
%
% subplot(5,1,2);
% imshow(ir);
% title('IR Image');
%
```

```
% subplot(5,1,3);
% imshow(veg4);
% title('Fused Image');
%
% subplot(5,1,4);
% h(1) = area(1:nBins, bHist,  'FaceColor', 'b'); hold on;
% h(2) = area(1:nBins, gHist,  'FaceColor', 'g'); hold on;
% h(3) = area(1:nBins, rHist, 'FaceColor', 'r'); hold on;
% plot( x, yBlue, 'Blue', x, yGreen, 'Green', x, yRed, 'Red');hold on;
% axis([0 xmax 0 ymax]);
% title('RGB image histogram');
%
% subplot(5,1,5);
% imhist(ir);

% hFig2(i) = figure(kk+2);
% kk = kk + 2;
%
subplot(3,2,1);
imshow(imOD);
title('Visual O-D Image');

subplot(3,2,2);
imshow(irOD);
title('IR O-D Image');

subplot(3,1,2);
imshow(veg2);
title('MNDVI Image');

subplot(3,1,3);
imshow(veg4);
title('TRF Image');


% Write out the original and processed images added by David L. Stone.
imwrite(im,(strcat(output1,'IMin', num2str(i),'.jpg'))); %Orignial Image
imwrite(ir,(strcat(output1,'IRin', num2str(i),'.jpg'))); %Orignial Image
imwrite(veg2,(strcat(output1,'MNDVI', num2str(i),'.jpg'))); %Oribnial image with MNDVI
overlay
imwrite(veg4,(strcat(output1,'MNDVI+J', num2str(i),'.jpg'))); %Originalimage iwht MNDVI+J
overlay%
imwrite(veg5,(strcat(output1,'MNDVI+NotJ', num2str(i),'.jpg'))); %Originalimage iwht
MNDVI+notJ overlay%
imwrite(veg_final,(strcat(output1,'MJMask', num2str(i),'.jpg'))); %MNDVI+J Mask
%imwrite(biColor,(strcat(output1,'vegGT', num2str(i),'.jpg'))); %ground truth
```

```
saveas(gcf, (strcat(outputAnnotateR_Hist,'Hist-Im', num2str(i),'.fig')));

% imwrite(im,(strcat(output3,'im', num2str(i),'.jpg'))); %Orignial Image
% imwrite(veg2,(strcat(output3,'MNDVI', num2str(i),'.jpg'))); %Oribnial image with MNDVI
overlay
% imwrite(veg4,(strcat(output3,'MNDVI-J', num2str(i),'.jpg'))); %Original image iwht
MNDVI+J overlay
% imwrite(veg_final,(strcat(output3,'MJMask', num2str(i),'.jpg'))); %MNDVI+J Mask

% Display: Can be changed to display any step of the process
%figure(2); imshow(im); title('original image');
%figure(3); imshow(veg2); title('MNDVI Overlay');
%figure(4); imsnow(veg4); title('Final MNDVI Image merged with J thermal vegetation region');
%figure(5); imshow(veg_final); title('MNDVI Mask merged with J thermal non vegetation
region ');
%figure(6); imshow(veg_final1); title(' not(J) not thermal non vegetation region ');
%figure(7); imshow(ir+J); title('IR Image + J thermal vegetation region');
%figure(8); imshow(ir+not(J)); title('IR Image + not(J)');

end
% % Print out the mndvi and thermalregion areas and the image total area

% xlswrite('vegDetStats.xls', imgArea, 'imgArea','B');
% xlswrite('vegDetStats.xls', MNDVIArea, 'MNDVIArea','C');
% xlswrite('vegDetStats.xls', trfArea, 'trfArea','D');
% xlswrite('vegDetStats.xls', mndviFP, 'mndviFP','E');
% xlswrite('vegDetStats.xls',trfFP,'trfFP','F');
fileid1 = fopen(strcat(output1, 'vegGTarea','.xls'),'w');
fprintf(fileid1, '%2.2f\n', vegGTarea);
fileid2 = fopen(strcat(output1, 'MNDVIArea','.xls'),'w');
fprintf(fileid2, '%2.2f\n', MNDVIArea);
fileid3 = fopen(strcat(output1, 'trfArea','.xls'),'w');
fprintf(fileid3, '%2.2f\n', trfArea);
fileid4 = fopen(strcat(output1, 'imgArea','.xls'),'w');
fprintf(fileid4, '%2.2f\n', imgArea);
fileid5 = fopen(strcat(output1, 'mndiFP','.xls'),'w');
fprintf(fileid5, '%2.2f\n', mndviFP);
fileid6 = fopen(strcat(output1, 'trfFP','.xls'),'w');
fprintf(fileid6, '%2.2f\n', trfFP);


fileid7 = fopen(strcat('thresholdData','.xls'),'w');
fprintf(fileid7, '%2.2f\n', TData,'TData', 'A');
fprintf(fileid7, '%2.2f\n', tLevel,'tLevel', 'E');

%xlswrite('thresholdData.xlsx', TData, 'TData', 'A');
```

```
%xlswrite('thresholdData.xlsx', tLevel, 'TData', 'E');


% fileid1 = fopen(strcat(output3, 'MNDVIArea','.txt'),'w');
% fprintf(fileid1, '%2.2f\n', MNDVIArea);
% fileid2 = fopen(strcat(output3, 'MNDVIJArea','.txt'),'w');
% fprintf(fileid2, '%2.2f\n', IRregion);
% fileid3 = fopen(strcat(output3, 'imgArea','.txt'),'w');
% fprintf(fileid3, '%2.2f\n', imgArea);
toc
% end of section added by David L. Stone
```

A.8 IR Region (IR_Region.m)

```
function J = IR_region(I, T1, T2)
% Function to find IR regions of similar temperature
% using a seed point
%   J = Logical output for region
%
%   I is the input image
%
%   David Stone
%
%   [J] = IR_region(I)

% initialize variables
%T1 = 100;
%T2 = 150;
thresVal = (T1 + T2)/2;
maxDist = (T2 - T1);
tfMean = true;
tfFillHoles = true;
tfSimplify = false;
test = 0;
v = 1;
idx = 1;
I = double(I);
% let I be the image (either NxM or NxMx3):

% the size:
s = size(I);
if length(s)==3
  %RGB
  I = reshape(I, s(1)*s(2), 3);
else
```

```
  I = I(:);
end

[m,n] = size(I);
% the random pixels:
while test == 0
   idx = randperm(m,1);
   v = I(idx,:); % grab the value of the pixel

   if and((v > T1), (v < T2))
      test = 1;
   else
      continue
   end
end
% and reshape back
I = reshape(I, s);

[r,c] = ind2sub(size(I),idx);

x1 = r; y1 = c;
%x1=191; y1=181;  % Seed point in region 1  need to add thresholding to identify seed point
%x1=102; y1=89;   % Seed point in region 2  need to add thresholding to identify seed point

J = regiongrowing(I,x1,y1,30);

%[P, J] = regionGrowing2(I, [r,c], thresVal, maxDist, tfMean, tfFillHoles,tfSimplify);

%J2 = regiongrowing(I,x2,y2,0.2);
```

A.9 Genetic Algorithm Clustering (GACluster.m)

```
clc;
clear;
close all;
ind=1;
%% Problem Definition
for ind = 1:20  % Do for twenty images
%
% Code to find the optimum threshold using a gynetic algorithm - Dave Stone
% May 2018
%
% Read in a standard MATLAB gray scale image.
%output_Region2Grow = 'C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\output_Region2Grow\';
```

```
%output_Region3Grow = 'C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\output_Region3Grow\';
output_Region4Grow_2 = 'C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation
Detection\output_Region4Grow_2\';
folder = 'C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\IRimage\';
output1='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\output1\'; %
Region 1 Output
output2='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\output2\'; %
Region 2 Output
output3='C:\Users\stoneda\Documents\MATLAB\Thesis\Vegetation Detection\output3\'; %
Region 3 Output
baseFileName = strcat('IRin',num2str(ind),'.jpg'); % modified by David Stone
fullFileName = fullfile(folder, baseFileName);
% Get the full filename, with path prepended.
fullFileName = fullfile(folder, baseFileName);
if ~exist(fullFileName, 'file')
    % Didn't find it there.  Check the search path for it.
    fullFileName = baseFileName; % No path this time.
    if ~exist(fullFileName, 'file')
        % Still didn't find it.  Alert user.
        errorMessage = sprintf('Error: %s does not exist.', fullFileName);
        uiwait(warndlg(errorMessage));
        return;
    end
end

% Read in the images
ir = (imread(fullFileName));
img = double(imread(fullFileName));
[s1,s2,s3]=size(img);
X1 = img(:,:,1);
X2 = img(:,:,2);
X3 = img(:,:,3);
X = [X1(:) X2(:) X3(:)]; % [(s1*s2)x3]

% Initialize variables

k = 2; % no. of clusters
outimg1 = zeros(s1,s2);
outimg2 = zeros(s1,s2);
outimg3 = zeros(s1,s2);

% Setup the threshold parameters and call the clusteirng cost function

[T1, T2, level, Iavg, thresholdvalue, binaryImage]= VThreshold(ir);
[sigmaW, TO1, TO2] = OptimumThreshold(T1, T2, X);
```

```
CostFunction=@(m) ClusteringCost2(m, X, TO1, TO2, k);     % Cost Function m = [3x2] cluster
centers

% Determine the Variables Matrix Size

VarSize=[k size(X,2)];  % Decision Variables Matrix Size = [k k-1]

nVar=prod(VarSize);     % Number of Decision Variables = 12

VarMin= repmat(min(X),1,k);      % Lower Bound of Variables [4x1] of[1x3] = [4x3]
VarMax= repmat(max(X),1,k);      % Upper Bound of Variables [4x1] of[1x3] = [4x3]

% Setting the Genetic Algorithms

ga_opts = gaoptimset('display','iter');

% running the genetic algorithm with desired options
[centers, err_ga] = ga(CostFunction, nVar,[],[],[],[],VarMin,VarMax,[],[],ga_opts);

% Refine the returned values

m=centers;

   % Calculate Distance Matrix
   g=reshape(m,k,k+1); % create a cluster center matrix(4(clusters) points in 3(features) dim
plane)=[4x3]
   d = pdist2(X, g); % create a distance matrix of each data points in input to each centers =
[(s1*s2)x4]

   % Assign Clusters and Find Closest Distances
   [dmin, index] = min(d, [], 2);
   % ind value gives the cluster number assigned for the input = [(s1*s2)x1]

   % Sum of Within-Cluster Distance
   WCD = sum(dmin);

   z=WCD; % fitness function contain sum of each data point to their corresponding center value
set (aim to get it minimum)
   % z = [1 x 1]

% Output the thresholded images
outimg =reshape(index,s1,s2);

% loop over the out image pixels and test for region k

   for i=1:s1
```

```
        for j=1:s2
            if outimg(i,j)== 1
                outimg1(i,j)= 1;
            elseif outimg(i,j)== 2
                outimg2(i,j)= 1;
%           elseif outimg(i,j)== 3
%               outimg3(i,j)= 1;
%           elseif outimg(i,j)== 4
%               outimg(i,j)= 0;
            end
        end
    end


outimg1 = logical(outimg1); %region 1 Image
outimg2 = logical(outimg2); %region 2 Image
% outimg3 = logical(outimg3); %region 3 Image
% outimg4 = logical(outimg3); %region 4 Image

% Write out to file the thresholded image

imwrite(outimg1,(strcat(output1,'irrg', num2str(ind),'.jpg'))); %write out region 1 Image
imwrite(outimg2,(strcat(output2,'irrg', num2str(ind),'.jpg'))); %write out region 2 Image
%imwrite(outimg3,(strcat(output3,'irrg', num2str(ind),'.jpg'))); %write out region 3 Image
%imwrite(outimg3,(strcat(output3,'irrg', num2str(ind),'.jpg'))); %write out region 3 Image

% Show the thresholded image

figure;imshow(outimg1);
figure;imshow(outimg2);
%figure;imshow(outimg3);
%figure;imshow(outimg3);

end
```

A.10 Clustering Cost Function (ClusteringCost2.m)

```
function z = ClusteringCost2(m, X, TO1, TO2,k)
%modified by Dave Stone May 2018
    WCD =0.0;
    % Calculate Distance Matrix
    g=reshape(m,k,k+1); % create a cluster center matrix(k(clusters) points in k-1(features) dim
plane)=[k x (k-1)]
    d = pdist2(X, g); % create a distance matrix of each data points in input to each centers =
[(s1*s2)xk]
%    while (TO1 <= d <= TO2)
%       dw = d;
%    end
    % Assign Clusters and Find Closest Distances
    [dmin, ~] = min(d, [], 2);
    % ind value gives the cluster number assigned for the input = [(s1*s2)x1]

    % Sum of Within-Cluster Distance
    for i = 1:1:length(dmin)
       if (TO1 <= dmin(i)) && (dmin(i) <= TO2)
          WCD = WCD + dmin(i);
       end
    end
    z=WCD; % fitness function contain sum of each data point to their corresponding center value
set (aim to get it minimum)

    % z = [1 x 1]
end
```

The Chapter 4 source codes for DeepFuseNet are as follows:

A.11 Bottleneck Feature Extractor (veg_bottleneck.py)
A.12 Autoencoder Feature Extractor (autoencode_feature.py)
A.13 Vegetation Finetuneing (veg_finetune.py)
A.14 Deep FusionNetwork (DeepFuseNet_Plants18.py)

A11. Bottleneck Feature Extractor (veg_bottleneck.py)

```
'''
"image classification models using limited data"
Bottleneck Feature Extractor
created by David Stone Sept 2018
In our setup, we:
- created a plants18/ folder
- created two sub folders /image and /IRimage each with below structure
- created train/, validation/ and test/ subfolders inside plants18/
- created grass/ trees/ and vegetation subfolders inside train/ and validation/ and test/
- put the grass pictures index 0-1125 in image/train/grass
- put the grass pictures index 721-1010 in image/validation/grass
- put the trees pictures index 0-1125 in data/train/trees
- put the trees pictures index 721-1010 in image/validation/trees
- put the vegetation pictures index 0-720 in data/train/vegetation
- put the vegetation pictures index 721-1010 in image/validation/vegetation

So that we have 1125 training examples for each class, and 290 validation examples for each class.
In summary, this is our directory structure:
```

~~~
```
plants18/
        images/
          train/
                trees/
                  image001.jpg
                  image002.jpg
                  ...
                vegetation/
                  image001.jpg
                  image002.jpg
                  ...
          validation/
                trees/
                  image001.jpg
                  image002.jpg
                  ...
                vegetation/
                  image001.jpg
                  image002.jpg
                  ...
```
'''
import h5py as h5
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D, ZeroPadding2D
from keras.layers import Activation, Dropout, Flatten, Dense, Input
from keras.applications.vgg16 import VGG16
from keras.models import model_from_json
from keras import applications
from keras.optimizers import RMSprop, SGD
from keras.utils.np_utils import to_categorical
from keras.utils import plot_model
from keras import backend as K
K.set_image_dim_ordering('tf')
import matplotlib.pyplot as plt
import math
import cv2
# dimensions of our images.
img_width, img_height = 224, 224
# path to the model weights file.
#weights_path = 'vgg16_weights.h5'
top_model_weights_path = 'bottleneck_fc_model.h5'
train_data_dir = 'plants18/images/train'
validation_data_dir = 'plants18/images/val'
nb_train_samples = 2250
nb_validation_samples = 690
epochs = 50
batch_size = 16
~~~

```
def save_bottleneck_features():
    # build the VGG16 network
    base_model = applications.VGG16(include_top=False, weights='imagenet')
    print("model loaded.")
    datagen = ImageDataGenerator(rescale=1. / 255)
    print(base_model.summary())
    plot_model(base_model, show_shapes = True, to_file = 'base_veg_model.png')
    generator = datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False)
    nb_train_samples = len(generator.filenames)
    num_classes = len(generator.class_indices)
    predict_size_train = int(math.ceil(nb_train_samples / float(batch_size)))
    bottleneck_features_train = base_model.predict_generator(
        generator, predict_size_train)
    np.save('bottleneck_features_train.npy', bottleneck_features_train)
    print("generator_train loaded.")
    generator = datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False)
    nb_validation_samples = len(generator.filenames)
    predict_size_validation = int(math.ceil(nb_validation_samples / float(batch_size)))
    bottleneck_features_validation = base_model.predict_generator(
        generator, predict_size_validation)
    np.save('bottleneck_features_validation.npy', bottleneck_features_validation)
    print("generator_val loaded.")
#    h5f= h5.File('veg_bottleNeck1.h5', 'w')
#    h5f.create_dataset('bottleneck_features_train', data=bottleneck_features_train)
#    h5f.create_dataset('bottleneck_features_validation', data=bottleneck_features_validation)
#   h5f.close()
def train_top_model():
    print("train_top_model...")
#    h5f= h5.File('veg_bottleNeck1.h5', 'r')
#    train_data= h5f['bottleneck_features_train'][:]
#    train_labels= np.array([grass]*(nb_train_samples//3), [tree]*(nb_train_samples//3),
[vegetation]*(nb_train_samples//3))
    # load the training data
    datagen_top = ImageDataGenerator(rescale=1./255)
    generator_top = datagen_top.flow_from_directory(
            train_data_dir,
            target_size=(img_width, img_height),
            batch_size=batch_size,
            class_mode='categorical',
            shuffle=False)
    nb_train_samples = len(generator_top.filenames)
    num_classes = len(generator_top.class_indices)
    # load the bottleneck features saved earlier
    train_data = np.load('bottleneck_features_train.npy')
    # get the class lebels for the training data, in the original order
    train_labels = generator_top.classes
    # convert the training labels to categorical vectors
    train_labels = to_categorical(train_labels, num_classes=num_classes)
    print("loaded train data...")
#   validation_data= h5f['bottleneck_features_validation'][:]
#   validation_labels= np.array([grass]*(nb_validation_samples//3), [tree]*(nb_validation_samples//3),
[vegetation]*(nb_train_samples//3))
#   h5f.close()
    # load the validation data
    generator_top = datagen_top.flow_from_directory(
            validation_data_dir,
            target_size=(img_width, img_height),
            batch_size=batch_size,
            class_mode='categorical',
            shuffle=False)
    nb_validation_samples = len(generator_top.filenames)
    validation_data = np.load('bottleneck_features_validation.npy')
    validation_labels = generator_top.classes
    validation_labels = to_categorical(validation_labels, num_classes=num_classes)
    print("loaded validation data...")
#   validation_data = np.load(open('bottleneck_features_validation', 'r'))
#   validation_labels = np.array(
#       [0] * (nb_validation_samples // 2) + [1] * (nb_validation_samples // 2))
    top_model = Sequential()
    top_model.add(Flatten(input_shape=train_data.shape[1:]))
    top_model.add(Dense(256, activation='relu'))
    top_model.add(Dropout(0.5))
    top_model.add(Dense(2, activation='softmax'))
    # compile the model with a SGD/momentum optimizer
    # and a very slow learning rate.
    sgd = SGD(lr=0.00001, decay=1e-6, momentum=0.95, nesterov=True)
    top_model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])
    print("compiled top_model")
    print(top_model.summary())
    plot_model(top_model, show_shapes = True, to_file = 'veg_model.png')
    history = top_model.fit(train_data, train_labels,
              epochs=epochs,
              batch_size=batch_size,
              validation_data=(validation_data, validation_labels))
    print("fit_generator")
    top_model.save('top_model.h5')
    top_model.save_weights('top_model_weights.h5')
    print("saved model and weights")
    (eval_loss, eval_accuracy) = top_model.evaluate(
        validation_data, validation_labels, batch_size=batch_size, verbose=1)
    print("[INFO] accuracy: {:.2f}%".format(eval_accuracy * 100))
    print("[INFO] Loss: {}".format(eval_loss))
    plt.figure(1)
    # summarize history for accuracy
    plt.subplot(211)
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    # summarize history for loss
    plt.subplot(212)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
save_bottleneck_features()
train_top_model()
```

A12. Autoencoder Feature Extractor

```
import h5py as h5
import numpy as np
import keras as K
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential
from keras.models import Model
from keras.layers import Dropout, Flatten, Dense
from keras.utils.np_utils import to_categorical
from keras import backend as K
K.set_image_dim_ordering('tf')
from keras.utils import plot_model
import matplotlib.pyplot as plt
import math
import cv2

# path to the model weights files.
weights_path = '../keras/examples/vgg16_weights.h5'
top_model_weights_path = 'plants18/hdf5/top_model_weights.h5'
# dimensions of our images.
img_width, img_height = 224, 224

train_data_dir = 'plants18/images/train'
validation_data_dir = 'plants18/images/val'
nb_train_samples = 2249
```

```
nb_validation_samples = 690
epochs = 150
batch_size = 16
num_classes = 2

# build the VGG16 network
model = applications.VGG16(weights='imagenet', include_top=False, input_shape = (224, 224, 3))
print('Model loaded.')

# build a classifier model to put on top of the convolutional model
top_model = Sequential()
top_model.add(Flatten(input_shape=model.output_shape[1:]))
top_model.add(Dense(256, activation='relu'))
top_model.add(Dropout(0.5))
top_model.add(Dense(2, activation='softmax'))

# note that it is necessary to start with a fully-trained
# classifier, including the top classifier,
# in order to successfully do fine-tuning
top_model.load_weights(top_model_weights_path)
model = Model(inputs= model.input, outputs= top_model(model.output))

# add the model on top of the convolutional base

print('Models added.')

# set the first 15 layers (up to the last conv block)
# to non-trainable (weights will not be updated)
for layer in model.layers[:16]:
    layer.trainable = False

# compile the model with a SGD/momentum optimizer
# and a very slow learning rate.
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(lr=5e-4, momentum=0.5),
              metrics=['accuracy'])
print("compiled model")

# prepare data augmentation configuration
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

nb_validation_samples = len(validation_generator.filenames)

validation_data = np.load('bottleneck_features_validation.npy')

validation_labels = validation_generator.classes
validation_labels = to_categorical(validation_labels, num_classes=num_classes)
print("train and val generator")

model.summary()

# fine-tune the model
history = model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size,
    verbose = 2)

print("finetuned model")

model.save('plants18/hdf5/veg_autoencode_model.h5')
model.save_weights('plants18/hdf5/veg_autoencode_model_weights.h5')

print("saved model and weights")

#(eval_loss, eval_accuracy) = model.evaluate(
#    history,
#    steps=nb_validation_samples // batch_size,
#    batch_size=16, verbose=1)

#print("[INFO] accuracy: {:.2f}%".format(eval_accuracy * 100))
#print("[INFO] Loss: {}".format(eval_loss))

plt.figure(1)

# summarize history for accuracy

plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

# summarize history for loss

plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

A.13 Vegetation Finetuning (veg_finetune.py)

```
# veg_finetune.py
import h5py as h5
import numpy as np
import keras
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential
from keras.models import Model
from keras.layers import Dropout, Flatten, Dense
from keras.utils.np_utils import to_categorical
from keras import backend as K(
K.set_image_dim_ordering('tf')
from keras.utils import plot_model
import matplotlib.pyplot as plt
import math
import cv2
# path to the model weights files.
weights_path = '../keras/examples/vgg16_weights.h5'
top_model_weights_path = 'plants18/hdf5/top_model_weights.h5'
# dimensions of our images.
img_width, img_height = 224, 224
train_data_dir = 'plants18/images/train'
validation_data_dir = 'plants18/images/val'
nb_train_samples = 2249
nb_validation_samples = 690
epochs = 30
batch_size = 32
num_classes = 2
# build the VGG16 network
model = applications.VGG16(weights='imagenet', include_top=False, input_shape = (224, 224, 3))
print('Model loaded.')
# build a classifier model to put on top of the convolutional model
```

```
top_model = Sequential()
top_model.add(Flatten(input_shape=model.output_shape[1:]))
top_model.add(Dense(256, activation='relu'))
top_model.add(Dropout(0.5))
top_model.add(Dense(2, activation='softmax'))
# note that it is necessary to start with a fully-trained
# classifier, including the top classifier,
# in order to successfully do fine-tuning
top_model.load_weights(top_model_weights_path)
model = Model(inputs= model.input, outputs= top_model(model.output))
# add the model on top of the convolutional base
print('Models added.')
# set the first 15 layers (up to the last conv block)
# to non-trainable (weights will not be updated)
for layer in model.layers[:16]:
    layer.trainable = False
# compile the model with a SGD/momentum optimizer
# and a very slow learning rate.
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(lr=5e-4, momentum=0.5),
              metrics=['accuracy'])
print("compiled model")
# prepare data augmentation configuration
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1. / 255)
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')
validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')
nb_validation_samples = len(validation_generator.filenames)
validation_data = np.load('bottleneck_features_validation.npy')
validation_labels = validation_generator.classes
validation_labels = to_categorical(validation_labels, num_classes=num_classes)
print("train and val generator")
model.summary()
# fine-tune the model
history = model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size,
    verbose = 2)
print("finetuned model")
model.save('plants18/hdf5/veg_finetune_model.h5')
model.save_weights('plants18/hdf5/veg_finetune_model_weights.h5')
print("saved model and weights")
#(eval_loss, eval_accuracy) = model.evaluate(
#    history,
#    steps=nb_validation_samples // batch_size,
#    batch_size=16, verbose=1)
#print("[INFO] accuracy: {:.2f}%".format(eval_accuracy * 100))
#print("[INFO] Loss: {}".format(eval_loss))
plt.figure(1)
# summarize history for accuracy
plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
# summarize history for loss
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

A14. Deep Fusion Network (DeepFuseNet_plants18.py)

```
# USAGE
# python3 DeepFuseNet_plants18.py --dataset1 ./plants18/images --dataset2 ./plants18/IRimages --model
./plants18/hdf5/veg_finetune_model.h5
#

# import the necessary packages

from sklearn import preprocessing as prep
from sklearn import model_selection
from sklearn import metrics
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from pyimagesearch.preprocessing import ImageToArrayPreprocessor
from pyimagesearch.preprocessing import AspectAwarePreprocessor
from pyimagesearch.datasets import SimpleDatasetLoader
from pyimagesearch.nn.conv import FCHeadNet
from imutils import paths
import numpy as np
import argparse
import os

import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import RMSprop
from keras.optimizers import SGD
from keras.applications import VGG16

#from config import imagenet_resnet_config as config
#from resnet101 import resnet101_model as ResNet
from keras.models import Model
from keras.layers import *
from keras.layers import Flatten, Input
from keras.layers.merge import concatenate
from keras.layers import Add, Dense, Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.backend import int_shape
from keras.utils import plot_model
from keras.callbacks import ModelCheckpoint

import tensorflow as tf
from keras.layers.core import Reshape
from keras import backend as K
K.set_image_dim_ordering('tf')
import matplotlib.pyplot as plt
import math
import json
import cv2
from IPython.display import clear_output
from livelossplot import PlotLossesKeras

import matplotlib
# Specifying the backend to be used before importing pyplot
# to avoid "RuntimeError: Invalid DISPLAY variable"
matplotlib.use('agg')
import matplotlib.pyplot as plt
import keras
import numpy as np

class TrainingPlot(keras.callbacks.Callback):

    # This function is called when the training begins
    def on_train_begin(self, logs={}):
        # Initialize the lists for holding the logs, losses and accuracies
        self.losses = []
        self.acc = []
        self.val_losses = []
        self.val_acc = []
        self.logs = []
        try:
            if first_call == True:
                self.testdone = 0
        except NameError:
            self.first_call = True
            self.testdone = 0

    # This function is called when the training ends
    def on_train_end(self, epoch):

         # Make sure there exists a folder called output in the current directory
         # or replace 'output' with whatever direcory you want to put in the plots
         try:
             if self.first_call == True:
                 self.first_call = False
                 self.testdone = 1
         except NameError:
             self.first_call = False
             self.testdone = 1

    # This function is called at the end of each epoch
    def on_epoch_end(self, epoch, logs={}):

        # Append the logs, losses and accuracies to the lists
        self.logs.append(logs)
        self.losses.append(logs.get('loss'))
        self.acc.append(logs.get('acc'))
        self.val_losses.append(logs.get('val_loss'))
        self.val_acc.append(logs.get('val_acc'))


        # Before plotting ensure at least 2 epochs have passed
        if len(self.losses) > 1:

            N = np.arange(0, len(self.losses))

            # You can chose the style of your preference
            # print(plt.style.available) to see the available options
            #plt.style.use("seaborn")

            # Plot train loss, train acc, val loss and val acc against epochs passed
            plt.figure()
            plt.plot(N, self.losses, label = "train_loss")
            plt.plot(N, self.acc, label = "train_acc")
            plt.plot(N, self.val_losses, label = "val_loss")
            plt.plot(N, self.val_acc, label = "val_acc")
            plt.title("Training Loss and Accuracy [Epoch {}]".format(epoch))
            plt.xlabel("Epoch #")
            plt.ylabel("Loss/Accuracy")
            plt.legend()

            # Make sure there exists a folder called output in the current directory
            # or replace 'output' with whatever direcory you want to put in the plots

            if self.testdone == 0:
                plt.savefig('output/Model1.png')

            if self.testdone == 1:
                plt.savefig('output/Model2.png')
      #           plt.savefig('output/Model2_Epoch-{}.png'.format(epoch))

      # construct the argument parse and parse the arguments
      ap = argparse.ArgumentParser()
      ap.add_argument("-d1", "--dataset1", required=True,
             help="path to input dataset1")
      ap.add_argument("-d2", "--dataset2", required=True,
             help="path to input dataset2")
      ap.add_argument("-m", "--model", required=True,
             help="path to output model")
      args = vars(ap.parse_args())

      # construct the image generator for data augmentation
      aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
             height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
             horizontal_flip=True, fill_mode="nearest")

      # grab the list of images that we'll be describing, then extract
      # the class label names from the image paths
      print("[INFO] loading images...")
      # setup image dataset 1
      imagePaths1 = list(paths.list_images(args["dataset1"]))
      classNames1 = [pt.split(os.path.sep)[-2] for pt in imagePaths1]
      classNames1 = [str(x) for x in np.unique(classNames1)]
      train_data_dir1 = 'plants18/images/train'
      val_data_dir1 = 'plants18/images/val'
      test_data_dir1 = 'plants18/images/test'
      # setup image dataset 2
      imagePaths2 = list(paths.list_images(args["dataset2"]))
      classNames2 = [pt.split(os.path.sep)[-2] for pt in imagePaths2]
      classNames2 = [str(x) for x in np.unique(classNames2)]
      train_data_dir2 = 'plants18/IRimages/train'
      val_data_dir2 = 'plants18/IRimages/val'
      test_data_dir2 = 'plants18/IRimages/test'
      # initialize the image preprocessors
      aap1 = AspectAwarePreprocessor(224, 224)
      iap1 = ImageToArrayPreprocessor()
```

```
# load the dataset from disk then scale the raw pixel intensities to
# the range [0, 1]
sdl1 = SimpleDatasetLoader(preprocessors=[aap1, iap1])
(data1, labels1) = sdl1.load(imagePaths1, verbose=500)
data1 = data1.astype("float") / 255.0
print("Data",data1,"labels",labels1)

# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX1, testX1, trainY1, testY1) = model_selection.train_test_split(data1, labels1,
        test_size=0.25, random_state=42)

# convert the labels from integers to vectors
trainY1 = prep.LabelBinarizer().fit_transform(trainY1)
testY1 = prep.LabelBinarizer().fit_transform(testY1)

ltrainX1 = len(trainX1)
ltestX1 = len(testX1)
batch_size = 64
img_height = 224
img_width = 224

input_imgen = ImageDataGenerator(rescale = 1./255,
                                 shear_range = 0.2,
                                 zoom_range = 0.2,
                                 rotation_range=5.,
                                 horizontal_flip = True)

test_imgen = ImageDataGenerator(rescale = 1./255)

# Here is the function that merges our two generators
# We use the exact same generator with the same random seed for both the y and angle arrays
def generate_generator_multiple(generator,dir1, dir2, batch_size, img_height,img_width):
    genX1 = generator.flow_from_directory(dir1,
                                          target_size = (img_height,img_width),
                                          class_mode = 'categorical',
                                          batch_size = batch_size,
                                          shuffle=False,
                                          seed=7)

    genX2 = generator.flow_from_directory(dir2,
                                          target_size = (img_height,img_width),
                                          class_mode = 'categorical',
                                          batch_size = batch_size,
                                          shuffle=False,
                                          seed=7)
    while True:
            X1i = genX1.next()
            X2i = genX2.next()
            yield [X1i[0], X2i[0]], X2i[1]  #Yield both images and their mutual label
```

```
inputgenerator=generate_generator_multiple(generator=input_imgen,
                                           dir1=train_data_dir1,
                                           dir2=train_data_dir2,
                                           batch_size=batch_size,
                                           img_height=img_height,
                                           img_width=img_height)

valgenerator=generate_generator_multiple(test_imgen,
                                          dir1=val_data_dir1,
                                          dir2=val_data_dir2,
                                          batch_size=batch_size,
                                          img_height=img_height,
                                          img_width=img_height)

testgenerator=generate_generator_multiple(test_imgen,
                                          dir1=test_data_dir1,
                                          dir2=test_data_dir2,
                                          batch_size=batch_size,
                                          img_height=img_height,
                                          img_width=img_height)


# load the VGG16 network, ensuring the head FC layer sets are left
# offerror: 'rand' was not declared in this scope

baseModel1 = VGG16(weights="imagenet", include_top=False,
   input_tensor=Input(shape=(224, 224, 3)))

input1 = baseModel1.input

#baseModel = ResNet(weights_path ='./resnet101_weights_tf.h5')
#include_top=False)     input_tensor = Input(shape=(224, 224, 3)))

# initialize the new head of the network, a set of FC layers
# followed by a softmax classifier
headModel1 = FCHeadNet.build(baseModel1, len(classNames1), 256)

# place the head FC model on top of the base model -- this will
# become the actual model we will train
model1 = Model(inputs=baseModel1.input, outputs=headModel1)

# loop over all layers in the base model and freeze them so they
# will *not* be updated during the training process
for layer in baseModel1.layers:
            layer.trainable = False

# compile our model
#(this needs to be done after our setting our
# layers to being non-trainable
print("[INFO] compiling model...")
opt = SGD(lr=0.001, momentum=0.7, nesterov=False)
```

```
model1.compile(loss="categorical_crossentropy", optimizer=opt,
          metrics=["accuracy"])

# train the head of the network for a few epochs (all other
# layers are frozen) -- this will allow the new FC layers to
# start to become initialized with actual "learned" values
# versus pure random
print("[INFO] training head...")
histories1 = []
histories1.append(
   model1.fit_generator(aug.flow(trainX1, trainY1, batch_size=batch_size),
            validation_data=(testX1, testY1), epochs=5,
        callbacks = [TrainingPlot()],
            steps_per_epoch=34, verbose=1))

#with open('plants18/json_out/file.json', 'w') as f:
#    json.dump(hist.history, f)

print("[INFO] compiling model1...")
opt = SGD(lr=0.0001, momentum=0.7, nesterov=False)
model1.compile(loss="categorical_crossentropy", optimizer=opt,
          metrics=["accuracy"])

# train the head of the network for a few epochs (all other
# layers are frozen) -- this will allow the new FC layers to
# start to become initialized with actual "learned" values
# versus pure random
print("[INFO] training head...")
histories1 = []
histories1.append(
   model1.fit_generator(aug.flow(trainX1, trainY1, batch_size=batch_size),
            validation_data=(testX1, testY1), initial_epoch = 6, epochs=10,
        callbacks = [TrainingPlot()],
            steps_per_epoch=ltrainX1/64, verbose=1))

# evaluate the network after initialization
print("[INFO] evaluating after training...")
predictions = model1.predict(testX1, batch_size=batch_size)
print(metrics.classification_report(testY1.argmax(axis=1),
          predictions.argmax(axis=1), target_names=classNames1))

# now that the head FC layers have been trained/initialized, lets
# unfreeze the final set of CONV layers and make them trainable
for layer in baseModel1.layers[15:]:
          layer.trainable = True

# for the changes to the model to take affect we need to recompile
# the model, this time using SGD with a *very* small learning rate
print("[INFO] re-compiling model...")
opt = SGD(lr=0.0001, momentum = 0.7, nesterov=False)
model1.compile(loss="categorical_crossentropy", optimizer=opt,
        metrics=["accuracy"])

# train the model again, this time fine-tuning *both* the final set
# of CONV layers along with our set of FC layers
print("[INFO] fine-tuning model...")
histories1 = []
histories1.append(
   model1.fit_generator(aug.flow(trainX1, trainY1, batch_size=batch_size),
        validation_data=(testX1, testY1), initial_epoch = 11, epochs=15,
       callbacks = [TrainingPlot()],
        steps_per_epoch=ltrainX1/64, verbose=1))
plt.close()

# evaluate the first network on the fine-tuned model
print("[INFO] evaluating first network after fine-tuning...")
predictions = model1.predict(testX1, batch_size=batch_size)
print(metrics.classification_report(testY1.argmax(axis=1),
        predictions.argmax(axis=1), target_names=classNames1))
print(model1.summary())

#Plot the results
#plt.figure(1)

# summarize history for accuracy
#plot_histories(histories1)

output1 = model1.output
#output1 = Reshape((224, 224,3))
#print(int_shape(output1))

#################

classNames2 = classNames1
# initialize the image preprocessors
aap2 = AspectAwarePreprocessor(224, 224)
iap2 = ImageToArrayPreprocessor()

# load the dataset from disk then scale the raw pixel intensities to
# the range [0, 1]
sdl2 = SimpleDatasetLoader(preprocessors=[aap2, iap2])
(data2, labels2) = sdl2.load(imagePaths2, verbose=500)
data2 = data2.astype("float") / 255.0
print("Data",data2,"labels",labels2)

# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX2, testX2, trainY2, testY2) = model_selection.train_test_split(data2, labels2,
        test_size=0.25, random_state=42)

# convert the labels from integers to vectors
trainY2 = prep.LabelBinarizer().fit_transform(trainY2)
```

```
testY2 = prep.LabelBinarizer().fit_transform(testY2)

ltrainX2 = len(trainX2)
batch_size = 64
img_height = 224
img_width = 224
#gen2 = ImageDataGenerator(horizontal_flip = True,
#                         vertical_flip = True,
#                         width_shift_range = 0.1,
#                         height_shift_range = 0.1,
#                         zoom_range = 0.1,
#                         rotation_range = 40)

# second network input

#grayimg = tf.get_variable("grayimg",[224,224,3])
#grayimg_1 = tf.image.rgb_to_grayscale(Input(shape=(224, 224, 3)))
#grayimg_2 = tf.image.rgb_to_grayscale(Input(shape=(224, 224, 3)))
#grayimg_3 = tf.image.rgb_to_grayscale(Input(shape=(224, 224, 3)))
#grayimg = tf.concat((grayimg_1, grayimg_2, grayimg_3), axis = -1)

baseModel2 = VGG16(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))

for layer in baseModel2.layers:
    layer.name = layer.name + str("_2")

input2 = baseModel2.input
output2 = baseModel2.output

#baseModel = ResNet(weights_path ='./resnet101_weights_tf.h5')
#include_top=False)     input_tensor = Input(shape=(224, 224, 3)))

# initialize the new head of the network, a set of FC layers
# followed by a softmax classifier
headModel2 = FCHeadNet.build(baseModel2, len(classNames2), 256)

# place the head FC model on top of the base model -- this will
# become the actual model we will train
model2 = Model(inputs=baseModel2.input, outputs=headModel2)

# loop over all layers in the base model and freeze them so they
# will *not* be updated during the training process
for layer in baseModel2.layers:
            layer.trainable = False

# compile our model
#(this needs to be done after our setting our
# layers to being non-trainable
print("[INFO] compiling model2...")
opt = SGD(lr=0.001, momentum = 0.7, nesterov=False)
```

```
model2.compile(loss="categorical_crossentropy", optimizer=opt,
        metrics=["accuracy"])

# train the head of the network for a few epochs (all other
# layers are frozen) -- this will allow the new FC layers to
# start to become initialized with actual "learned" values
# versus pure random
print("[INFO] training head...")
histories2 = []
histories2.append(
  model2.fit_generator(aug.flow(trainX2, trainY2, batch_size=batch_size),
         validation_data=(testX2, testY2), epochs=1,
      callbacks = [TrainingPlot()],
         steps_per_epoch=ltrainX1/64, verbose=1))

# compile our model
#(this needs to be done after our setting our
# layers to being non-trainable
print("[INFO] compiling model2...")
opt = SGD(lr=0.0001, momentum = 0.7, nesterov=False)
model2.compile(loss="categorical_crossentropy", optimizer=opt,
        metrics=["accuracy"])

# train the head of the network for a few epochs (all other
# layers are frozen) -- this will allow the new FC layers to
# start to become initialized with actual "learned" values
# versus pure random
print("[INFO] training head...")

histories2 = []
histories2.append(
  model2.fit_generator(aug.flow(trainX2, trainY2, batch_size=batch_size),
        validation_data=(testX2, testY2), initial_epoch = 2, epochs=2,
     callbacks = [TrainingPlot()],
        steps_per_epoch=ltrainX1/64, verbose=1))

# evaluate the network after initialization
print("[INFO] evaluating after initialization...")
predictions = model2.predict(testX2, batch_size=batch_size)
print(metrics.classification_report(testY2.argmax(axis=1),
        predictions.argmax(axis=1), target_names=classNames2))

# now that the head FC layers have been trained/initialized, lets
# unfreeze the final set of CONV layers and make them trainable
for layer in baseModel2.layers[15:]:
        layer.trainable = True

# for the changes to the model to take affect we need to recompile
# the model, this time using SGD with a *very* small learning rate
print("[INFO] re-compiling model...")
opt = SGD(lr=0.00001, momentum = 0.7, nesterov=False)
```

```
model2.compile(loss="categorical_crossentropy", optimizer=opt,
        metrics=["accuracy"])

# train the model again, this time fine-tuning *both* the final set
# of CONV layers along with our set of FC layers
print("[INFO] fine-tuning model...")
histories2 = []
histories2.append(
model2.fit_generator(aug.flow(trainX2, trainY2, batch_size=batch_size),
              validation_data=(testX2, testY2), initial_epoch = 3,epochs=3,
           callbacks = [TrainingPlot()],
              steps_per_epoch=ltrainX1/64, verbose=1))

# evaluate the second network on the fine-tuned model
print("[INFO] evaluating after fine-tuning second model...")
predictions = model2.predict(testX2, batch_size=batch_size)
print(metrics.classification_report(testY2.argmax(axis=1),
        predictions.argmax(axis=1), target_names=classNames2))
plt.close()

model2.get_layer(name='flatten').name='flatten_1'
print(model2.summary())

#Plot the results
#plt.figure(2)
#plot_histories(histories2)

output2 = model2.output
#output2 = Reshape((224, 224,3))
#print(int_shape(output2))

#####################################################################

#trainX1 = np.reshape(trainX1, (-1, 2217,2))
#trainX2 = np.reshape(trainX2, (-1, 2217,2))
#trainX1 = trainX1.reshape(trainX1.shape[1:])
#testX1 = testX1.transpose()
#trainX2 = trainX2.reshape(trainX2.shape[1:])
#testX2 = testX2.transpose()
#mergemodel = keras.layers.Add()([model1.output, model2.output])
mergemodel = Concatenate()([output1, output2])

print("line 1")
#flat1 = Flatten()(mergemodel)

print("line 2")
# summarize layers
# plot graph

hidden1 = Dense(3, activation='relu')(mergemodel)
```

```
output = Dense(3, activation ='softmax')(hidden1)
print("line 3")

Finalmodel = Model(inputs=[input1, input2], outputs=output)
#Finalmodel = Model(inputs=model1.inputs+model2.inputs, outputs=output)

# summarize layers
print(Finalmodel.summary())
# plot graph
plot_model(Finalmodel, to_file='multiple_inputs.png')

# compile our model
#(this needs to be done after our setting our
# layers to being non-trainable
print("[INFO] compiling merged model...")

opt = SGD(lr=0.000001,  momentum = 0.7, nesterov=False)
Finalmodel.compile(loss="categorical_crossentropy", optimizer=opt,
        metrics=["accuracy"])
print(Finalmodel.summary())

print("line 4")
# train the head of the network for a few epochs (all other
# layers are frozen) -- this will allow the new FC layers to
# start to become initialized with actual "learned" values
# versus pure random
print("[INFO] training merged head...")
print(classNames1)
print(len(classNames1))
print(len(testX1))
print(len(trainX1))
print(len(testX2))
print(len(trainX2))
#Y=np.append(trainY1,trainY2, axis=0)
#Ytest=np.append(testY1,testY2,axis=0)

#history = Finalmodel.fit_generator(inputgenerator2,
#         validation_data=(testgenerator2), initial_epoch = 7,epochs=10,
#       validation_steps=ltrainX2 // 64,
#       steps_per_epoch=ltrainX2 // 64, verbose=1)
print(trainX1[0].shape)
print(trainX2[0].shape)
print(trainY1[0].shape)
print(testX1[0].shape)
print(testX2[0].shape)
print(testY1[0].shape)

#history = Finalmodel.fit_generator([trainX1,trainX2], trainY1,
#            initial_epoch = 4,epochs=4,
#            callbacks = callback_tensorboard("plants18/logs/run_c"),
#            validation_data = ([testX1,testX2], testY1))
```

```
            #  use_multiprocessing=True,
#classNames1=['tree', 'vegetation']

#history = Finalmodel.fit_generator(aug.flow([trainX1,trainX2], trainY1, batch_size=batch_size),
#                   validation_data=(testX1, testY1), initial_epoch = 3,epochs=3,
#                 callbacks = callback_tensorboard("plants18/logs/run_b"),
#                   steps_per_epoch=ltrainX1/64, verbose=1)

history=Finalmodel.fit_generator(inputgenerator,
                    steps_per_epoch=ltrainX1/batch_size,
                    epochs = 1,
                    validation_data = valgenerator,
                    validation_steps = ltestX1/batch_size,
                    use_multiprocessing=True,
                    shuffle=False)
print(Finalmodel.summary())

#Plot the results
plt.figure(3)

# summarize history for accuracy

plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('merged model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')

# summarize history for loss

plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('merged model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss', 'val_loss'], loc='upper left')
#plt.show()
plt.savefig('mergedmodel.png', bbox_inches='tight')
print("line 5")

# save the model to disk
print("[INFO] serializing model...")
#Finalmodel.save(args["mergemodel"])
Finalmodel.save('plants18/hdf5/plants18_finetune_model.h5')
Finalmodel.save_weights('plants18/hdf5/plants18_finetune_model_weights.h5')

# evaluate the merged network on the fine-tuned model
print("[INFO] evaluating after final merge...")
```

```
n_classes = 3
print("line 6")
predictions = Finalmodel.predict([testX1,testX2], batch_size=batch_size)
con_mat = tf.confusion_matrix(labels=['grass', 'tree', 'vegetation'], predictions=predictions,
num_classes=n_classes, dtype=tf.int32, name=None)

with tf.Session():
  print('Confusion Matrix: \n\n', tf.Tensor.eval(con_mat,feed_dict=None, session=None))

print(metrics.classification_report([testY1.argmax(axis=1), testY2.argmax(axis=1)],
         predictions.argmax(axis=1), target_names=[classNames1, classNames2]))
```

References

[1] H. Anderson, A. Treptow, and T. Duckett, "Self-Localization in Non-Stationary Environments Using Omnidirectional Vision", *Robotics and Autonomous Sys*. Vol. 55, pp 541 – 551, 2007.

[2] C Cauchois, E. Brassart, C. Drocourt, and P. Vasseur, "Calibration of the Omnidirectional Vision Sensor: SYCLOP", in *Proc. IEEE Intl. Conf. on Robotics and Automation*, Vol. 2, pp 1287 – 1292, 1999.

[3] S. Baker, and S. Nayar, "A Theory of Single-Viewpoint Catadioptric Image Formation ", *Intl. J. of Computer Vision*, 35(2), pp 1 – 22, 1999.

[4] G. Caron, E. Mouaddib, and E. Marchand, "3D Model based tracking for omnidirectional vision: A new spherical approach", *Robotics and Autonomous Sys*. Vol. 60, pp 1056-1068, 2012.

[5] C. Geyer and K. Daniilidis, "Catadioptric projective geometry", *Intl J. Computer Vision*, 43, pp 223-243, 2001.

[6] L. Puig, J. Bermudez, P. Sturm, and J. Guerrero, "Calibration of omnidirectional cameras in practice: A comparison of methods", *Computer Vision and Image Understanding*, Vol. 116, No. 1, pp 120-147, 2012.

[7] D. Scaramuzza, A. Martinelli, and R. Siegwart, “A Flexible Technique for Accurate Omnidirectional Camera Calibration and Structure from Motion”, in Proc. *IEEE Intl. Conf. of Vision Systems (ICVS'06)*, New York, 2006.

[8] D. Scaramuzza, A. Martinelli, and R. Siegwart, “A Toolbox for Easy Calibrating Omnidirectional Cameras”, *in Proc. IEEE Intl. Conf. Intelligent Robots and Sys. (IROS 2006),* Beijing China, 2006.

[9] C. Mei., and P. Rives, “Single View Point Omnidirectional Camera Calibration from Planar Grids”, *Proc. IEEE Intl. Conf. Robotics and Automation*, 2007.

[10] H. Tang, Y. Liu, “Automatic Simultaneous Extrinsic-Odometric Calibration for Camera-Odometry System”, IEEE Sensors J., vol. 18, no. 1, pp. 348 - 355, 2018.

[11] L. Puig, Y. Bastanlar, P. Sturm, J. Guerrero, and J. Barreto, “Calibration of Central Catadioptric Cameras Using a DLT-Like Approach”, *Intl. J. Computer Vision*, 93: pp. 101–114, 2011.

[12] Y. Motai and A. Kosaka, "Hand-Eye Calibration Applied to Viewpoint Selection for Robotic Vision", *IEEE Trans. Industrial Electronics*, Vol. 55, Issue 10, pp.3731-3741, 2008.

[13] Y. Motai and A. Kosaka, "SmartView: Hand-Eye Robotic Calibration for Active Viewpoint Generation and Object Grasping", *Proc. IEEE Intl. Conf. Robotics and Automation*, Vol. 3, pp.2183-2190, 2001.

[14] X. Peng, M. Bennamoun, Q. Wang, Q. Ma, and Z; Xu, “A Low-Cost Implementation of a 360° Vision Distributed Aperture System”, *IEEE Trans. Circuits and Systems for Video Technology*, vol. 25, no. 2, pp. 225-238, 2015.

[15] E. Benli, J. Ralph, Y. Motai, “Dynamic 3-D Reconstruction of Human Targets via an Omni-Directional Thermal Sensor”, *IEEE Sensors J.,* vol., 18 no. 17, pp. 7209-7221, 2018.

[16] D. Strelow, J. Mishler, D. Koes, and S. Singh, “Precise Omnidirectional Camera

Calibration", in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 689-694, December, 2001.

[17] Z. Yan, L. Zhao, and H. Wanbao, "A Survey of Catadioptric Omnidirectional Camera Calibration", *Intl. J. Information Technology and Computer Science*, Vol.5 (3), pp 13-20, 2013.

[18] H. Duan, and W. Yihong, "A Calibration Method for Paracatadioptic Camera from Sphere", *Pattern Recognition Letters*, Vol.33 (6), pp 677-684, 2012.

[19] C. Geyer, and K. Daniilidis, "Paracatadioptic camera calibration", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.24(5), pp 687-695, 2002.

[20] D. Santana-Cedrés, L. Gomez, M. Alemán-Flores, A. Salgado, J. Esclarín L. Mazorra, L. Alvarez, "Estimation of the Lens Distortion Model by Minimizing a Line Reprojection Error", IEEE Sensors J., vol. 17, no. 9, pp. 2848 - 2855, 2017

[21] Z. Zhang, T. Tan, K. Huang, and Y. Wang, "Practical Camera Calibration from Moving Objects for Traffic Scene Surveillance", *IEEE Trans. Circuits and Systems for Video Technology*, vol. 23, no. 3, pp. 518-533, 2013.

[22] R. Summan, G. Dobie, G. West, S. Marshall, C. MacLeod, and S. G. Pierce, "The Influence of the Spatial Distribution of 2-D Features on Pose Estimation for a Visual Pipe Mapping Sensor", *IEEE Sensors J.*, vol. 17, no. 19, pp. 6313-6321, 2018.

[23] X. Ying and Z. Hu, "Catadioptric camera calibration using geometric invariants", *IEEE Trans. Pattern Analysis and Machine Intelligence,* Vol. 26, Issue 10, pp. 1260 – 1271, 2004.

[24] X. Deng, F. Wu, Y. Wu, F. Duan, L. Chang, and H. Wang, "Self-calibration of hybrid central catadioptric and perspective cameras", *Computer Vision and Image Understanding*, Vol.116

(6), pp. 715-729 2012.

[25] S. Ramalingam, P. Sturm and S. Lodha "Generic Self Calibration of Central Cameras", *Computer Vision and Image Understanding*, Vol.114, pp. 210-219, 2010.

[26] L. Krüger, and C. Wöhler, "Accurate chequer board corner localization for camera calibration", *Pattern Recognition Letters*, Vol.32 (10), pp.1428-1435, 2011.

[27] J. Barreto, and H. Araujo "Geometric properties of central catadioptric line images and their application in calibration", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.27 (8), pp.1327-33, 2005.

[28] J. Barreto "A unifying geometric representation for central projection systems", *Computer Vision and Image Understanding*, Vol.103 (3), pp.208-217, 2006.

[29] A. Bonarini, P. Aliverti, and M. Lucioni. "An Omnidirectional Vision Sensor for Fast Tracking for Mobile Robots", *IEEE Trans Intelligent Transportation Systems,* VOL. 49(3), 2000.

[30] A. Agrawal, S. Ramalingam. "Single image calibration of multi-axial imaging systems." *Proc. Computer Vision and Pattern Recognition (CVPR)*, pp 1399-1406, 2013.

[31] L. Zhou, "A New Minimal Solution for the Extrinsic Calibration of a 2d LIDAR and a Camera Using Three Plane-Line Correspondences", *IEEE Sensors Journal* Vol. 14, no. 2, pp. 442–454, 2014.

[32] J. Y. Bouguet. (2013, Oct. 10). *Camera Calibration Toolbox* [Online]. Available: http://www.vision.caltech.edu.bouguetj/calib_doc

[33] J. Kannala, J. Heikkilä and S. Brandt. "Geometric camera calibration". Wiley Encyclopedia of Computer Science and Engineering, 2008.

[34] J. Kannala and S. Brandt. "A generic camera model and calibration method for conventional,

wide-angle and fish-eye lenses". *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 8, 2006.

[35] W. Wong, C. Loo and W. Lim, "Optical Approach Based Omnidirectional Thermal Visualization", *Intl. J. Image Processing*, Vol. 4(3). pp. 263, 2010.

[36] O. Romain, T. Ea., and P. Garda, "Multispectral omnidirectional vision sensor: design, calibration, and utilization", *Optical Engineering*, Vol.46 (10), 12 pages, 2007.

[37] T. Mouats, N. Aouf, L. Chermak, and M. A. Richardson, "Thermal stereo odometry for UAVs," *IEEE Sensors J.*, vol. 15, no. 11, pp. 6335–6347, Nov. 2015.

[38] S. You, R. Wei, A. Robles-Kelly, "A Multiview Light Field Camera for Scene Depth Estimation from a Single Lens", *Proc. AML arXiv*, 2016.

[39] S. You, R. Tan, R. Kawakami, Y. Mukaigawa and K. Ikeuchi, "Water drop Stereo", *in Computing Research Repository arXiv*, vol. 1604.00730, 2016.

[40] G. Agrawal, K. Lewis, K. Chung, C-H. Huang S. Parashar, and C. Bloebaum, "Intuitive Visualization of Pareto Frontier for Multi-Objective Optimization in n-Dimensional Performance Space", *American Institute of Aeronautics and Astronautics, Technical Report*, 2004.

[41] E. Zitler M. Laumans, and S. Bleuler, "A Tutorial on Evolutionary Multiobjective Optimization", *Springer, Metaheuristics for Multiobjective Optimization, Lecture Notes Economics and Mathematical Systems*, Vol.535, pp. 3-37, 2004.

[42] H. Anderson, A. Treptow, T. Duckett, "Self-Localization in Non-Stationary Environments Using Omni-directional Vision", *Robotics and Autonomous Systems* vol. 55, pp. 541 – 551, 2007.

[43] G. Lopez-Nicolas, J. Guerrero, and C. Sagues, "Multiple homographies with

omnidirectional vision for robot homing", *Robotics and Autonomous Systems*, vol. 58 no. 6, pp. 773-783, 2010.

[44] W. Lui, and R. Jarvis, "Eye-full tower: A GPU-based variable multi-baseline omnidirectional stereovision system with automatic baseline selection for outdoor mobile robot navigation", *Robotics and Autonomous Systems*, vol. 58 no. 6, pp. 747-761, 2010.

[45] J. Lim, and N. Barnes, "Estimation of the epipole using optical flow at antipodal points", *Computer Vision and Image Understanding*, vol. 114 no. 2, pp. 245-253, 2010.

[46] D. Stone, G. Shaw, Y. Motai, "Direct Spherical Calibration of Omnidirectional Far Infrared Camera System", *submitted to IEEE Sensors Journal*, 2018.

[47] D. Scaramuzza, A. Martinelli, and R. Siegwart, "A Robust Descriptor for Tracking Vertical Lines in Omnidirectional Images and its Use in Mobile Robotics", *International Journal on Robotics Research,* vol. 28, issue 2, 2009.

[48] S. Aksoy, "Spatial Techniques for Image Classification", *Signal and Image Processing for Remote Sensing,* New York: Taylor and Francis, pp. 491-513, 2006.

[49] O. Helwich, C. Wiedemann, "Object Extraction from High-Resolution Multi-Sensor Image Data", *Image and Signal Processing for Remote Sensing Int. Soc. Opt. Eng.* vol. 3871, pp. 284-295, 1999.

[50] D. Landgrebe, "Information Extraction Principles and Methods for Multispectral and Hyperspectral Image Data", Purdue University, 1998.

[51] Y. J. Kaufman and D. Tanré, "Atmospherically resistant vegetation index (ARVI) for EOS-MODIS," *IEEE Trans. Geosci. Remote Sensing*, vol. 30, pp. 261–270, Mar. 1992.

[52] H. Q. Liu and A. R. Huete, "A feedback modification of the NDVI to minimize canopy background and atmospheric noise," *IEEE Trans. Geosci. Remote Sensing*, vol. 33, pp. 457–

465, Mar. 1995.

[53] M. Verstraete, B. Pinty, “Designing Optimal Spectral Indices for Remote Sensing Applications”, *IEEE Trans. Geoscience and Remote Sensing,* vol. 34 no. 5, pp. 1254-1265, 1996.

[54] C. Ünsalan, K. Boyer, “Linearized Vegetation Indices Based on a Formal Statistical Framework”, *IEEE Trans. Geosci. Remote Sensing,* vol. 42, no. 7, pp. 1575-`585, 2004.

[55] M. Bunkei, Y. Wei Y C. Jin, O. Yuichi, Q. Guoyu, "Sensitivity of the enhanced vegetation index (EVI) and normalized difference vegetation index (NDVI) to topographic effects: a case study in high-density cypress forest." *Sensors,* vol. 7 no. 11, pp. 2636-2651, 2007.

[56] M. Bradley, S. M. Thayer, A. Stentz, P. Rander, “Vegetation detection for mobile robot navigation”, CMU-RI-TR-04-12, Robotics Institute, Carnegie Mellon University, 2004

[57] D. Bradley, R. Unnikrishnan, and J. Bagnell, "Vegetation Detection for Driving in Complex Environments", Robotics *and Automation, 7 IEEE International Conference*, 2007.

[58] P. D’Odorico, A. Gonsamo, A. Damm, and M. Schaepman, “Experimental Evaluation of Sentinel-2 Spectral Response Functions for NDVI Time-Series Continuity”, IEEE Trans. Geoscience and Remote Sensing, vol.51, no. 3, pp. 1336-1348, 2013.

[59] K. Bhoyar, O. Kakde,” Color Image Segmentation Based on JND Color Histogram”, *Intl Jrnl. Image Processing (IJIP)*, Vol. 3, no. 6, pp. 283-292, 2010.

[60] A. Briggs, C. Detweiler, P. Mullen, D. Sharstein, “Scale Space Features in 1D” Omnidirectional *Images*, *OMNIVIS*, pp. 115 – 126, 2004.

[61] A. Pretto, E. Minegatti, J. Yoshiaki, U. Ryuichi, and A. Tamio, “Image similarity based on discrete wavelet transform for robots with low-computational resources”, *Robotics and Autonomous Systems*, vol. 58 no. 7, pp. 879-888. 2010.

[62] M. Taiana, J. Santos, J. Gaspar, J. Nascimento, A. Bernardino, P. Lima, “Tracking objects with generic calibrated sensors: An algorithm based on color and 3D shape features”, *Robotics and Autonomous Systems*, vol. 58 no. 6, pp. 784-795, 2010.

[63] A. Bagnell, D. Bradley, D. Silver, B. Sofman, “Learning Rough-Terrain Autonomous Navigation”, *IEEE Robotics and Automation Magazine,* vol. 17 no. 2, pp. 74-84, 2010.

[64] A. Rankin, A. Huertas, L. Matthies, M. Bajracharya., C. Assad., S. Brennan, P. Bellutta, G. Sherwin, “Unmanned ground vehicle perception using thermal infrared cameras”, SPIE *Defense, Security, and Sensing*, International Society for Optics and Photonics, pp. 804503-804503, 2011.

[65] D. Wolf, G. Sukhatme, “Activity-Based Semantic Mapping of an Urban Environment”, *Springer Tracts in Advanced Robotics,* vol. 39 pp 321-329, 2008.

[66] A. Mishra, Y, Alimonos, “Visual Segmentation of Simple Objects for Robots”, *Proc. Robotics: Science and Systems* vol. 7 pp.30-38, 2012.

[67] D. Nguyena, L. Kuhnert, K. Kuhnert, “Structure overview of vegetation detection. A novel approach for efficient vegetation detection using an active lighting system”, *Robotics and Autonomous Systems,* vol. 60 pp.498–508, 2012.

[68] T. Hoang, P. Duong, N. Van, D. Viet and T. Vinh, “Multi-Sensor Perceptual System for Mobile Robot and Sensor Fusion-based Localization”, *International Conf. Control, Automation, and Information Sciences*, pp. 259-264, 2012.

[69] C. Cimander, M. Carlsson, and C. F. Mandenius, “Sensor fusion for on-line monitoring of yoghurt fermentation,” *Jrnl. Biotechnol.*, vol. 99, pp. 237–248, 2002.

[70] A. Ross and A. Jain, “Information fusion in biometrics,” *Pattern Recognit. Lett*. vol. 24, pp. 2115–2125, 2003.

[71] M. A. K. Jaradat and R. Langari, “A hybrid intelligent system for fault detection and sensor fusion,” *Appl. Soft. Comput*. vol. 9, pp. 415–422, 2009.

[72] C. Aliustaoglu, H. M. Ertunc, and H. Ocak, “Tool wear condition monitoring using a sensor fusion model based on fuzzy inference system,” *Mech. Syst. Signal Process*., vol. 23, pp. 539–546, 2009.

[73] S. Tangjitsitcharoen and C. Rungruang, “In-process monitoring and estimation of tool wear on CNC turning by applying multi-sensor with back propagation technique,” *Adv. Materials Res*., vol. 291, pp. 3036–3043, 2011.

[74] Q. Cheng, P. K. Varshney, J. H. Michels, and C. M. Belcastro, “Distributed fault detection with correlated decision fusion,” *IEEE Trans. Aerosp. Electron. Syst*., vol. 45, pp. 1448–1465, 2009.

[75] K. Liu, and S. Huang, “Integration of Data Fusion Methodology and Degradation Modeling Process to Improve Prognostics”, *IEEE Trans. Automation Science and Engineering*, vol. 13 no. 1, pp. 344-354, 2016.

[76] N. Ahmed, E. Sample, and M. Campbell, “Bayesian Multi-Categorical Soft Data Fusion for Human–Robot Collaboration”, *IEEE Trans. Robotics*, vol. 29, no. 1, pp. 189-206, 2013.

[77] O. Beyca, P. Rao, Z. Kong, S. Bukkapatnam, and R. Komanduri, “Heterogeneous Sensor Data Fusion Approach for Real-time Monitoring in Ultra-Precision Machining (UPM) Process Using Non-Parametric Bayesian Clustering and Evidence Theory”, *IEEE Trans. Automation Science and Engineering*, vol. 13 no. 2, pp. 1033-1044, 2016.

[78] B. Schäfer, C. Armbrust, T. Föhst, and K. Berns, *“*The Application of Design Schemata in Off-Road Robotic*s”, IEEE Intelligent Transportation Systems Magazine,* pp. 1-27, 2013.

[79] Z. Kang, J. Yang, and R. Zhong, “A Bayesian-Network-Based Classification Method

Integrating Airborne LiDAR Data with Optical Images", *IEEE Jrnl. Of Selected Topics in Applied Earth Observations and Remote Sensing,* vol. 10, no. 4, pp. 1651-1661, 2017.

[80] D. Dutta, K. Wang, E. Lee, A. Goodwell, D.K. Woo, D. Wagner, and P. Kumar, "Characterizing Vegetation Canopy Structure Using Airborne Remote Sensing Data", IEEE *Trans. Geoscience and Remote Sensing*, vol. 55, no. 2, 2017.

[81] C.-H. Chen, Y. Yao, D. Page, B. Abidi*, Senior Member, IEEE*, A. Koschan*, Member, IEEE*, and M. Abidi, "Heterogeneous Fusion of Omnidirectional and PTZ Cameras for Multiple Object Tracking", *IEEE Trans. Circuits and Systems for Video Technology*, vol. 18, no. 8, 2008.

[82] N. Robertson, J. Letham, "Contextual Person Detection in Multi-modal Outdoor Surveillance", *20th European Signal Processing Conference (EUSIPCO 2012),* pp. 1930-1934, 2012.

[83] D. Andújar, J. Dorado, C. Fernández-Quintanilla, and A. Ribeiro, "An Approach to the Use of Depth Cameras for Weed Volume Estimation", *Sensors,* vol. 16, no 972, pp.1-11, 2016.

[84] V. Poulain, J. Inglada, M. Spigai, J.-Y. Tourneret, and P. Marthon, "High-Resolution Optical and SAR Image Fusion for Building Database Updating", *IEEE Trans, Geoscience and Remote Sensing*, vol. 49, no. 8, pp. 2900-2910, 2011.

[85] M Jiayi, C. Chen, C. Li, J. Huang, "Infrared and visible image fusion via gradient transfer and total variation minimization", *Information Fusion,* Vol. 31, pp. 100-109, 2016.

[86] H. Li, H. Qiu, Z. Yu ⇑, Y. Zhang," Infrared and visible image fusion scheme based on NSCT and low-level visual features", *Infrared Physics and Technology*, vol. 76, pp. 174-184, 2016.

[87] M. Mancas, B. Gosselin, M, Benoit. Segmentation using a region growing thresholding.

Image Processing: Algorithms and Systems IV. vol. 5672. pp. 388-398, 2006.

[88] B. Russell, A. Torralba, K. Murphy, W. T. Freeman, LabelMe: a database and web-based tool for image annotation . International Journal of Computer Vision, vol. 77, is. 1-3, pp. 157-173, 2007.

[89] S. Prasad, H. Wu, J. E. Fowler, "Compressive Data Fusion for Multi-Sensor Image Analysis," *in Proceedings of the International Conference on Image Processing, Paris, France*, October 2014, pp. 5032-5036.

[90] A Zaitunah et al, "Normalized difference vegetation index (NDVI) analysis for land cover types using Landsat 8 oli in besitang watershed, Indonesia", *IOP Conf. Ser.: Earth Environ. Sci*. 126 012112, 2018.

[91] T. Sankey, J. McVay, T. Swetnam, M. McClaran, P. Heilman, and M. Nichols, "UAV hyperspectral and LIDAR data and their fusion for arid and semi-arid land vegetation monitoring", *Remote Sensing in Ecology and Conservation*, 4 (1):20–33, 2018.

[92] L. Leite, L. Carvalho, F. Silva, "Change Detection in Forests And Savannas Using Statistical Analysis Based on Geographical Objects", *Bol. Ciênc. Geod., sec. Artigos, Curitiba*, v. 23, no2, pp.284 - 295, 2017

[93] L. Reymondina, A. Jarvis, A. Perez-Uribe, J. Touval, K. Argotea, J. Rebetezc, E. Guevaraa, M. Mulligane "A methodology for near real-time monitoring of habitat change at continental scales using MODIS-NDVI and TRMM", submitted to *Remote Sensing of Environment*, 2010

[94] R. Carlà, L. Santurri, L. Bonora, C. Conese, "Multi-temporal burnt area detection methods based on a couple of images acquired after the fire event", *5th Intl. Wildland Fire Conf.*, 2011.

[95] S. Aksoy, "Spatial Techniques for Image Classification", *Geoscience and Remote Sensing, IEEE Trans. on* 46 (7), 2097-2111, 2008.

[96] O. Helwich, C. Wiedemann, “Object Extraction from High-Resolution Multisensor Image Data”, Technical University of Munich, *Third Intl. Conf. Fusion of Earth Data*, 2000.

[97] D. Landgrebe, “Information Extraction Principles and Methods for Multispectral and Hyperspectral Image Data”, *Purdue University*, 1998.

[98] M. Bunkei, Y. Wei Y C. Jin, O. Yuichi, Q. Guoyu, “Sensitivity of the Enhanced Vegetation Index (EVI) and Normalized Difference Vegetation Index (NDVI) to Topographic Effects: A Case Study in High-Density Cypress Forrest”, *Sensors 2007*, 7(11), 2636-2651 2007.

[99] M. Bradley, S. M. Thayer, A. Stentz, P. Rander, “Vegetation detection for mobile robot navigation”, *CMU-RI-TR-04-12, Robotics Institute Carnegie Mellon University*, 2004.

[100] D. Bradley, R. Unnikrishnan, and J. Bagnell, "Vegetation Detection for Driving in Complex Environments", *Robotics Institute*, 52., available http://repository.cmu.edu/robotics/52, 2007.

[101] K. Bhoyar, O. Kakde O., “Color Image Segmentation Based on JND Color Histogram”, *Intl. Jrnl. of Image Processing (IJIP),* Vol 3, Issue 6, pp 283-292, 2010.

[102] A. Briggs, C. Detweiler, P. Mullen, D. Sharstein, “Scale Space Features in 1D Omnidirectional Images”, *OMNIVIS*, pp 115 – 126, 2004.

[103] A. Pretto, E. Minegatti, J. Yoshiaki, U. Ryuichi, U., & A. Tamio, “Image similarity based on discrete wavelet transform for robots with low-computational resources”, *Robotics and Autonomous Systems*, 58(7), 879-888, 2010.

[104] M. Taiana, J. Santos, J. Gaspar, J. Nascimento, A. Bernardino, & P. Lima, “Tracking objects with generic calibrated sensors: An algorithm based on color and 3D shape features”, *Robotics and Autonomous Systems*, 58(6), 784-795, 2010.

[105] X. Zhang, J. Zhang, C. Li, C. Cheng, L. Jiao, H. Zhou, “Hybrid Unmixing Based on

Adaptive Region Segmentation for Hyperspectral Imagery", IEEE Trans. Geoscience and Remote Sensing, vol. 56, no. 7, pp. 3861 - 3875, 2018.

[106] A. Rankin, A. Huertas, L. Matthies, M. Bajracharya., C. Assad., S. Brennan, P. Bellutta, G. Sherwin, "Unmanned ground vehicle perception using thermal infrared cameras", *Jet Propulsion Lab, California Institute of Technology and General Dynamics Robotic Systems, SPIE*, 2011.

[107] D. Wolf, G. Sukhatme, "Activity-Based Semantic Mapping of an Urban Environment", *Robotic Embedded Systems, USC*, *Autonomous Robots*, 19(1):53–65, 2004.

[108] A. Mishra, Y, Alimonos, "Visual Segmentation of Simple Objects for Robots", *Proc. Robotics Science and Systems, pp. 30-38, 2011.*

[109] J. Cruz Ortiz, "Visual servoing for an omnidirectional mobile robot using the neural network - Multilayer perceptron" *IEEE Workshop on Engineering Applications (WEA)*, pp. 1-6, 2012.

[110] M. Figueiredo, S. Botelho, P. Drews and C. Haffele, "Spatial and Perceptive Mapping Using Semantically Self-Organizing Maps Applied to Mobile Robots", *IEEE Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), Brazilian*, pp. 245-250, 2012.

[111] S. Kai-Tai, and C. Chang, "Reactive navigation in dynamic environment using a multi-sensor predictor", *IEEE Trans. Systems, Man, and Cybernetics, Part B: Cybernetics*, 29(6), pp. 870 – 880, 1999.

[112] K. Dongshin, S. Jie, M. Sang, J. Rehg, A. Bobick, "Traversability Classification using Unsupervised On-line Visual Learning for Outdoor Robot Navigation", *Proc. - IEEE Intl. Conf. Robotics and Automation, 2006.*

[113] S. Chang, S. Kai-Tai, “Environment Prediction for a Mobile Robot in a Dynamic Environment”, *IEEE Trans. Robotics and Automation*, 13(6), pp 862-872, 1997.

[114] A. Coates, A. Karpathy, A. Ng,” Emergence of Object-Selective Features in Unsupervised Feature Learning”, *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 2012.

[115] A. Coates, H. Lee, A. Ng, “An Analysis of Single-Layer Networks in Unsupervised Feature Learning”, *Fourteenth Intl Conf. on Artificial Intelligence and Statistics (AISTAT)*, pp 215 - 223, 2011.

[116] E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks”, *SCIENCE,* Vol 313, pp 504-507, 2006.

[117] P. Vincent, H. Larochelle, Y. Bengio, P. Manzagol, “Extracting and Composing Robust Features with De-noising Autoencoders”, *Machine Learning*, 2008.

[118] H. Lee, R. Battle, A. Raina, A. Ng, “Efficient Sparse Coding Algorithms”, NIPS, 2006.

[119] S. Hao, W. Wang, Y. Ye, T. Nie, L. Bruzzone, “Two-Stream Deep Architecture for Hyperspectral Image Classification”, IEEE Trans. Geoscience and Remote Sensing, vol. 56, no. 4, pp. 2349 - 2361, 2018.

[120] W. Song, S. Li, L. Fang, T. Lu, “Hyperspectral Image Classification With Deep Feature Fusion Network “, IEEE Trans. Geoscience and Remote Sensing, vol. 56, no. 6, pp. 3173 - 3184, 2018.

[121] I. Halatci, C. Brooks, and K. Iagnemma. "Terrain classification and classifier fusion for planetary exploration rovers.", *Aerospace Conference, IEEE.*, pp 1-11, 2007.

[122] X. Zhuo, F. Fraundorfer, F. Kurz, P. Reinartz, “Building Detection and Segmentation Using a CNN with Automatically Generated Training Data”, IEEE Trans. Geoscience and Remote Sensing, pp. 3461 – 3464, 2018.

[123] C. Craye, D. Filliat_ and J-F Goudou, “Exploration Strategies for Incremental Learning of Object-Based Visual Saliency”, Development and Learning and Epigenetic Robotics (ICDL-EpiRob)”, *Joint IEEE International Conference on. IEEE*, pp 13 – 18, 2015.

[124] C. Siagian, L. Itti, "Biologically Inspired Mobile Robot Vision Localization,”, *IEEE Trans on Robotics,* vol.25, no.4, pp.861-873, 2009.

[125] A. Kelly, A Stentz, O Amidi, M Bode, "Toward reliable off road autonomous vehicles operating in challenging environments." *Intl. J. of Robotics Research*, 25.5-6, pp 449-483, 2006.

[126] F. Gao., Y. Xun, J. Wu, G. Bao, & Y. Tan, “Navigation line detection based on robotic vision in natural vegetation-embraced environment", *Image and Signal Processing (CISP), 2010 3rd International Congress on, IEEE* Vol. 6., pp 2596 - 2600 2010.

[127] M. Veres, G. Lacey, and G. W. Taylor, "Deep Learning Architectures for Soil Property Prediction", *Computer and Robot Vision (CRV), 12th IEEE Conference,* pp 8-15, 2015.

[128] J. Nagi, F. Ducatelle, G. Di Caro, D. Cireşan, U. Meier, A. Giusti, & L.M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition", *Signal and Image Processing Applications (ICSIPA), IEEE Intl. Conf.,* pp 342-347, 2011.

[129] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg and L Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge”, *Intl. Jnl. of Computer Vision (IJCV),* Vol.13 (3), pp. 211-252, 2015.

[130] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” *in Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR),* 2016.

[131] A. Rosebrock, “Deep Learning for Computer Vision with Python”, *PyImageSearch,* 2017.

[132] F. Chollet, “Deep Learning with Python”, *Manning Publishing Co.*, 2018.

[133] K. Simonyan, A. Zisserman “Very Deep Convolutional Networks for Large-Scale Image Recognition”, *arXiv:1409.1556v6 [cs.CV],* 2015.

[134] K. He, X. Zhang, S. Ren, J. Sun, “Deep Residual Learning for Image Recognition”, *arXiv:1512.03385v1 [cs.CV],* 2015.

[135] D. Stone, G. Shaw, Y. Motai, A. Aved, “Vegetation Segmentation for Sensor Fusion of Omnidirectional Far-infrared and Visual Stream”, *submitted to IEEE Jnl. Of Selected Topics in Appllied Earth Observations and Remote Sensing (JSTARS),* 2018.

[136] X. Zhuo, F. Fraundorfer, F. Kurz, P. Reinartz, “Building Detection and Segmentation Using a CNN with Automatically Generated Training Data”, *Geoscience and Remote Sensing, IEEE Trans. On*. pp. 3461 – 3464, 2018.

[137] Y. Yu, P. Zhong, Z. Gong, “Balanced data driven sparsity for unsupervised deep feature learning in remote sensing images classification”, *Geoscience and Remote Sensing, IEEE Trans. On*, pp. 668-671, 2017.

[138] K. Chen, K. Fu, X. Gao, M. Yan, X. Sun, H. Zhang, “Building extraction from remote sensing images with deep learning in a supervised manner”, *Geoscience and Remote Sensing, IEEE Trans. On*, pp.1672-1675, 2017.

David Lee Stone | Curriculum Vitae

PhD in Electrical and Computer Engineering (GPA:3.5) Aug.2010–Dec.2018
Virginia Commonwealth University Richmond, VA

Research Area

Robotic Perception, Machine Learning, GPU Computing, Parallel Computing.

Industry Experience

- **Senior Robotics Engineer, Dahlgren, VA**
  Naval Surface Warfare Center, Marine Corps Warfighting Lab Sep.2009–Jan.2019
- **Lead Systems Engineer III, Liberty Lake, WA, U.S.**
  Isothermal Systems Research, Inc. (SprayCool), Jul.2004–Aug.2009
- **President, and Chief Technology Officer, Coure d' Alene, ID, U.S.**
  Unmanned Systems Technology Lab, Inc. (USTLAB), 2001-Dec.2004
- **Senior Systems Engineer, Bayview, ID, U.S.**
  Science Applications International Corp. (SAIC), Aug.1994-Dec.2002
- **Engineering Duty Officer, Naval Surface Warfare Officer, U.S. Navy, Various Ships**
  Active Duty in the U.S. Navy as a Commissioned Officer, Jun.1973-Aug.1994
- **Electronics Technician, U.S. Navy, Various Ships and Bases**
  Active Duty Electronics Technician, Enlisted, Oct.1966-Jun.1973

Education

Academic Qualifications……………………………………………………………………

- **Virginia Commonwealth University Richmond, VA, U.S.**
  Electrical and Computer Engineering, PhD Aug.2010–Jan.2019
- **Massachusetts Institute of Technology, Cambridge, MA, U.S.**
  Ocean Engineer OE, Naval Architect, Mechanical Engineering, M.S. Jun.1979–Aug.1982
- **Purdue University, West Lafayette, IN, U.S**
  Electrical Engineering, Minor in Computer Architecture B.S. Sep.1969–May.1973

Teaching Assistant…………………………………………………………………………

1982 Spring/Summer *Working as a TA for then OCEAN ENGINEERING SHIP DESIGN Class*

Technical and Personal skills

- **Professional Certifications:** Professional Engineer Commonwealth of Virginia, Certified Interconnect Designer
- **Programming Languages:** C/C++/python, Matlab, Assembly, CUDA, , ORCAD Circuit Capture and Layout, Solid Modeling and Manufacturing - SolidWorks, Inventor, BobCAD/CAM

- **Other Proficiencies:** Rapid Prototyping - Fused Deposition Manufacturing, PCB Proto Machining, Acoustic Analysis, Acoustic Quieting.
- **Computer Architecture:** CPU, GPU, On-chip interconnection, Programmable Gate Arrays.
- **Statistical Skills:** multi-variable linear regression, Design of Experiments.
- **Other Training:**
  - National Robotic Engineering Center (NREC) Robotics Course (2012)
  - Marine Corps Acculturation Program (MCAP) (2010)
  - DAWIA Level III Certification – Systems Planning, Research, Development and Engineering – Systems Engineering (2010)
  - SPIE Defense & Security - GPS Technology, FPGA Programming Fundamentals (2007)
  - Kepner Trego, Problem Solving & Decision Making (2005)
  - System Engineering for Program Managers, UCLA, (2005)
  - Basic and Advanced MEMS Courses, Sandia National Lab (2002)
  - CID Certification, IPC Interconnect Designer Certification Course (2001)
  - Digital Signal Processing, (1998)
  - MATLAB basics and MATLAB graphics, (1998)
  - Experimental Design Techniques, (1992)
  - Basic Statistical Process Improvement Techniques, (1991)
  - U.S. Navy Submarine Officers School, (1990)
  - Leadership Management Training, (1990)
  - Basic Engineering Officers School, (1982)
  - Electronics Maintenance Officer School, (1976)
  - Communications Officer School, (1974)
  - Salvage and Diving School, U.S. Navy (1973)
  - Electronics Technician “A”, “B”, “C” schools (Radars), U.S. Navy (1967)

Publications

D. Stone and D. Caufield, “NAVO Chirp E9901A Data Analysis And Review, Report No. 3007-2”, *Caufield Engineering Services Ocean Data Equipment Corporation, for Naval Oceanographic Office*, 4 April 2001.

D. Stone and D. Caufield, “Variance Analysis in NAVO Chirp E9901A Data, Report No. 3007-2A”, *Caufield Engineering Services Ocean Data Equipment Corporation, for Naval Oceanographic Office*, 1 May 2001.

D. Stone and D. Caufield, “NAVO Watergun Cruise 261199 Data Analysis and Review, Report No. 3007-3”, *Caufield Engineering Services Ocean Data Equipment Corporation, for Naval Oceanographic Office*, 4 May 2001.

J. Cranney and D. Stone, “Applications of Modeling Manta Ray motion, Kinematics, Energy, and Endurance of robotic Simulator”, *Unmanned Systems Technology Laboratory, Inc. (USTLAB), 2nd International Symposium on Aqua Bio-Mechanisms ISABMEC 2003*, 14-17 September 2003.

D. Stone, J. Cranney , "The Application of SMA Spring Actuators to a Lightweight Modular Compliant Surface Bio-inspired Robot", *Unmanned Systems Technology Laboratory, Inc. (USTLAB), Robert Liang, Dr. Minoru Taya, University of Washington, SPIE Smart Structures and Materials Conference*, 16 March 2004.

D. Stone, J. Cranney, "The Development of a Lightweight Modular Compliant Surface Bio-inspired Robot", *USTLAB, Inc., SPIE Defense and Security Conference, Unmanned Ground Vehicle Technology VI*, 13 April 2004.

A.D. Johnston, D.L. Stone, T. Cader & D.A. Locklear, "SprayCool™ Compact Server for military shipboard environments", *Isothermal Systems Research Inc., THERMES 2007*, 7-10 Jan 2007.

A. Johnston, D. Stone, T. Cader, Ph.D., "SprayCool® Command Post Platform for Harsh Military Environments", *SprayCool, Inc., 24th IEEE SEMI-THERM* 2008.

D. Stone, E. Ballew, S. Zimmerman, "Impact of SprayCool® Cooling Technology Applied to Power Electronics on Shipboard Cooling System Infrastructure", *Electric Machine Technology Symposium (EMTS 2008)*, 12-13 August 2008.

L. Baraniecki, J. Vice, J. Brown, J. Nichols, Anthrotronix, Inc., D. Stone, D. Dawn, Marine Corps Warfighting Lab,, "Intuitive Robotic Operator Control (IROC): Integration of Gesture Recognition with an Unmanned Ground Vehicle and Heads up Display", *NDIA, Ground Vehicle Systems Engineering and Tehcnology Symposium (GVSETS),* pp. 1-7, 2016.

C. Kawatsu, F. Koss, A. Gillies, A. Zhao, J. Crossman, B. Purman; Soar Technology; D. Stone; D. Dahn, Marine Corp Warfighting Lab, "Gesture Recognition for Robotic Control using Deep Learning", *NDIA, Ground Vehicle Systems Engineering and Technology Symposium (GVSETS),* 2017.

D. Stone, G. Shaw, Y. Motai, A. Aved, " Vegetation Segmentation for Sensor Fusion of Omnidirectional Far-infrared and Visual Stream", *Journal of Selected Topics in Applied Earth Observations and Remote Sensing, In press, DOI: 10.1109/JSTARS.2019.2891518.*

D. Stone, G. Shaw, Y. Motai, "Direct Spherical Calibration of Omnidirectional Far Infrared Camera System", submitted *IEEE Sensors Journal, 2019.*

D. Stone, S. Ravi, Y. Motai, "Vegetation Detection through Deep Sensor Fusion of Omnidirectional Far-infrared and Visual Stream", *IEEE Transactions on Geoscience and Remote Sensing, submitted IEEE TGRS-2019-00023, 2019.*